UNIVERSIDAD PONTIFICIA COMILLAS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO SUPERIOR INDUSTRIAL

PROYECTO FIN DE CARRERA

PRIMERAS ETAPAS EN EL DISEÑO DE UN ESTABILIZADOR DE VUELO PARA HELICÓPTEROS A ESCALA

Parte I: Sistema de Adquisición y Almacenamiento de Datos

AUTOR: **MIGUEL ÁNGEL TORROBA CAÑAS**

MADRID, JUNIO 2005

Parte I Memoria *4*

Capítulo 1 Introducción *5*

1 Precedentes **5**

2 Motivación del proyecto **6**

3 Objetivos **7**

4 Metodología **8**

5 Visión general **10**

Capítulo 2 Unidad de Medida Inercial (IMU) *12*

1 Introducción **12**

1.1 Objetivo de la Unidad de Medida Inercial 12

1.2 Breves apuntes sobre los sensores empleados 13

1.2.1 Acelerómetros 13

1.2.2 Giróscopos 15

2 Hardware de la Unidad de Medida Inercial **16**

2.1 Características principales 16

2.2 Descripción del hardware 17

2.3 Montaje y conexión 21

2.4 Modificación de los filtros anti-aliasing 21

3 Software de la Unidad de Medida Inercial **24**

3.1 Análisis 24

3.2 Modificaciones a nivel de software 33

3.2.1 Mainloop.c 33

3.2.2 Config.h 33

3.2.3 Ppm.h 34

3.2.4 Servo.c 35

3.2.5 Makefile 35

4 Resultados parciales obtenidos con la IMU **36**

5 Futuras ampliaciones del Sistema de Medida **37**

5.1 Otras variables que convendría medir 37

Capítulo 3 Sistema de Almacenamiento de Datos (SAD) *38*

1 Introducción **38**

2 Etapas de desarrollo **39**

2.1 Etapa 1 - De la idea hasta los componentes. 40
2.2 Etapa 2 - Desarrollo con Eagle®. 45
2.2.1 Introducción de componentes. 45
2.2.2 El conector para tarjetas tipo SD. 45
2.2.3 Conexionado eléctrico. 46
2.2.4 Disposición real de componentes. 47
2.2.5 Tablas y resúmenes. 49
2.3 Etapa 3 – Fuse Bits y Bootloader. 52
2.4 Etapa 4 – Adaptación de la librería a las necesidades del proyecto 55
2.4.1 Descripción 55
2.4.2 Consideraciones de memoria 56
2.4.3 Opciones de compilación 57
2.5 Etapa 5 – Comunicación con la Unidad de Medida Inercial. 59
2.5.1 El buffer circular. 59
2.5.2 Programa main. 61
2.5.3 Función *initialize_media()* de la biblioteca FlashFile. 63
2.5.4 Función *fcreate* de la biblioteca FlashFile. 63
2.5.5 Función *fprintf* de la biblioteca FlashFile. 64
2.5.6 Función *fflush* de la biblioteca FlashFile. 64
2.5.7 Función *fclose* de la biblioteca FlashFile. 65

3 Resultados 65

4 Recursos empleados 67

4.1 Eagle 68
4.2 AVR Dude de Win AVR 68
4.3 AVR Studio 4 68
4.4 Kargador y bootloader 68
4.5 FlashFile 70
4.6 CodeVision 70

5 Otras posibles aplicaciones 72

Capítulo 4 Integración de IMU y SAD. 73

1 La caja apropiada 73

2 Ubicación de los módulos 73

3 Sistema de aislamiento 74

4 Sistema de sujeción 75

5 Panel de mandos 76

6 Proceso de montaje paso-a-paso 77

7 Ejes 80

Capítulo 5 Receptor 82

Capítulo 6 Conclusiones. 85

Bibliografía 87

Anexo I: Guía básica sobre manejo de AVR ATMEL Atmega 88

1 Características generales 88

2 Programación 89

3 Arquitectura del microprocesador 89

4 Utilidades 90

4.1 Puertos I/O. Funciones alternativas. 90

4.2 Memoria. 91

4.3 Vía SPI –Serial Periferial Interface-. 92

4.4 USART –Universal Syncronous Asyncronous Reciever Transmiter- 96

4.5 Timers 100

4.6 PWM. 103

4.7 Interrupciones. 105

4.8 Otros recursos no utilizados en este proyecto 106

Anexo II: Código del Sistema de Adquisición de Datos 107

Anexo III: Código modificado de la Unidad de Medida Inercial 117

Mainloop.c 117

Config.h 132

Ppm.h 136

Servo.c 146

Makefile 156

Anexo IV: Características del conector/zócalo para tarjetas SD 165

Parte II Estudio Económico 168

Capítulo 1 Coste del Proyecto 169

Parte III Manual de usuario 172

Parte I MEMORIA

Capítulo 1 INTRODUCCIÓN

La presente memoria pretende mostrar con toda la claridad y detalle posible la motivación con que arrancó este proyecto, sus fases de elaboración, la metodología de trabajo empleada y los recursos utilizados para su desarrollo. Además se realiza con la idea de que los resultados y conclusiones alcanzadas, así como el "know how" adquirido sobre las tecnologías empleadas, puedan ser utilizados en el futuro desarrollo de otros proyectos, sean o no continuación directa de este.

1 Precedentes

El transporte aéreo es la forma de transporte moderno que más rápidamente se ha desarrollado. Aunque los Hermanos Wright hicieron el primer vuelo de la historia en un aparato más pesado que el aire en Carolina del Norte en el año 1903, no fue hasta después de la I Guerra Mundial cuando el transporte aéreo alcanzó un lugar destacado en todos los países.

Tras la II Guerra Mundial los aviones comerciales recibieron un mayor impulso gracias a la evolución de los propulsores de los aviones (desde la hélice hasta los turborreactores) y desde entonces la industria de la aviación no ha dejado de desarrollarse tecnológicamente.

El siguiente paso en la evolución de esta industria viene de la mano de los sistemas que electrónica avanzada, comunicaciones y sistemas informáticos para aumentar la eficiencia y seguridad del transporte. Uno de

los avances más conocidos en este campo es el Piloto Automático. Son sistemas de navegación en los que los pilotos introducen sus destinos y el aparato actúa para llegar al mismo. Este sistema está muy desarrollado en los aviones, pero en pañales si hablamos de helicópteros.

El siguiente paso a dar en el desarrollo de la aviación es la creación de sistemas de control que conviertan los helicópteros en Vehículos Aéreos Autónomos. La tecnología y los conocimientos necesarios para este propósito ya están disponibles y la investigación en este campo viene de la mano del ejército americano. A parte, varias universidades en el mundo están desarrollando proyectos para crear estos sistemas aplicándolos a helicópteros a escala, algunos ejemplos posibles son el MIT –Massachussets Institute of Technology- y ahora el IIT –Instituto de Investigación Tecnológica de la Universidad Pontificia de Comillas- que empieza su andadura en el campo de los Helicópteros No Tripulados con este proyecto que será la base de futuros desarrollos.

2 Motivación del proyecto

El proyecto "Primeras etapas para el desarrollo de un estabilizador de vuelo para helicópteros a escala" surgió, al principio, con la intención de la construcción de Vehículo Aéreo Autónomo (en adelante VAA). Para llegar a la exitosa consecución de este amplio requisito hubiera sido necesario, además de un equipo humano más amplio que el que participante en esta aventura, un periodo de tiempo mucho más dilatado y unos medios en absoluto justificables para un Proyecto Fin de Carrera (en adelante PFC). Por estos motivos se redujo la pretensión inicial de crear un VAA en su

totalidad para hacer converger los objetivos hacia tareas de creación de una infraestructura que sirviera de base para la continuidad futura del objetivo principal. Ajustando el tiempo, los recursos y el equipo humano disponible a dichas tareas, así el título del Proyecto global queda como se ha mencionado anteriormente y se divide en dos partes bien diferenciadas, una para cada uno de los proyectandos implicados:

- Sistema de Adquisición y Almacenamiento de Datos.
- Análisis de los Datos y Estimación del Estado.

Las aplicaciones de un sistema como el comentado serán tan variadas como variadas sean las situaciones en las que se pueda aplicar una acción aérea; desde el reconocimiento de una zona en búsqueda de algo concreto (bancales de malas hierbas, por ejemplo) hasta la supervisión de instalaciones industriales.

3 Objetivos

Con todo lo expuesto anteriormente podríamos resumir los objetivos del proyecto como se hace a continuación:

El objetivo general del proyecto es establecer la infraestructura necesaria para facilitar futuros estudios sobre la identificación de parámetros y el control de un helicóptero a escala.

El objetivo general de esta parte es diseñar y montar un Sistema de Adquisición y Almacenamiento de Datos que sirva de recurso para la segunda parte del proyecto, lo que conlleva los siguientes objetivos parciales:

- Medir con la mayor precisión posible las aceleraciones lineales y angulares mediante los sensores adecuados (acelerómetros y giróscopos) integrándolos en una Unidad de Medida Inercial –en inglés Inertial Measurement Unit, IMU-. Todo ello a un precio razonable.
- Almacenar dichas variables con un formato sencillo en un sistema de reducido tamaño y peso y lo más versátil posible para facilitar su reutilización en otras tareas de almacenamiento de datos.
- Montar el sistema completo formado por las dos unidades anteriores en una caja para darles robustez y compactación.

4 Metodología

La primera parte, "Sistema de Adquisición y Almacenamiento de Datos", consiste en el diseño y la construcción de un módulo que, colocado en el helicóptero a escala o en cualquier otro objeto en movimiento, sea capaz de medir y almacenar las aceleraciones lineales y las velocidades angulares del mismo, a partir de las cuales, en la segunda parte -"Análisis de los Datos y Estimación del Estado"- se pueda obtener mediante una serie de filtros y análisis el estado del helicóptero en el aire, es decir, su posición y orientación angular en los tres ejes del espacio.

Aunque ambas partes se presentan por separado, el trabajo se ha realizado de forma conjunta por ambos proyectandos durante la duración integra del PFC porque permite, sobre todo, una mayor calidad de los resultados y agilizar ambas partes -principalmente la primera, cuya

finalización es vital para el comienzo de la segunda- obteniendo un conocimiento más amplio sobre el tema del proyecto.

La primera parte, de la que soy autor oficial, es la que se expondrá en este documento.

Todas las fases tienen una estructura más o menos típica:

- Búsqueda de información. La mayor parte de la documentación (manuales de componentes, información sobre la IMU que se compró,…) para esta parte del proyecto ha sido extraída de Internet.
- Diseño o adaptación de los sistemas.
- Montaje.
- Programación en C.
- Pruebas de funcionamiento de los sistemas antes de que puedan ser utilizados en el helicóptero. Primero comprobación del funcionamiento en tierra y luego en el aire.

5 Visión general

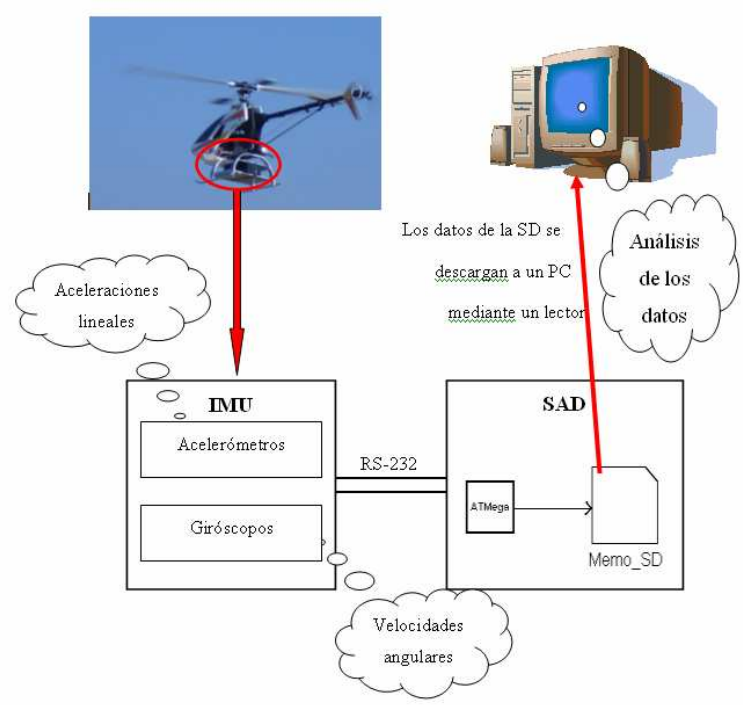


Ilustración 1 – Visión general del proyecto

Observando la Ilustración 1 se puede comprender de un solo vistazo las intenciones de esta parte del proyecto. Se tiene un helicóptero a escala en el aire tripulado desde tierra, en el que se ha dispuesto un equipo electrónico compuesto por:

1- Una Unidad de Medida Inercial (IMU), destinada a seguir el movimiento del mismo por medio de tres acelerómetros y tres giróscopos.

2- Un Sistema de Almacenamiento de Datos (SAD), utilizado para almacenar los datos lanzados por la anterior en un medio físico como es una tarjeta comercial SD, por lo tanto, ambos sistemas crean en conjunto un Sistema de Adquisición y Almacenamiento de Datos (SAAD).

La consecución de unos datos repetitivos y de calidad para facilitar una futura identificación de los parámetros que caracterizan la respuesta dinámica del helicóptero frente a cambios en las entradas (órdenes del piloto) y en las "entradas" externas (por ejemplo, ráfagas de viento) es el objetivo principal de la segunda parte del proyecto.

Capítulo 2 UNIDAD DE MEDIDA INERCIAL (IMU)

1 Introducción

1.1 Objetivo de la Unidad de Medida Inercial

La determinación del estado de un cuerpo en 3D (posición y orientación espacial) pasa por la medida y análisis de una serie de variables inerciales, a saber aceleraciones lineales y velocidades angulares en los tres ejes del espacio. Tarea que se realiza mediante dos clases de sensores: acelerómetros y giróscopos.

En un primer momento se buscaron los sensores más convenientes con el ánimo de realizar esta unidad por nosotros mismos. Vistas las dificultades que se planteaban se siguió una solución alternativa: la búsqueda de una placa que integrase estos dispositivos, además del software que permita su utilización. Tras varios modelos encontrados que, aunque de gran calidad y versatilidad, resultaban excesivamente caros, se localizó uno que estaba dentro de las posibilidades económicas de este proyecto. Se adquirió en Rotomotion LLC por 324,95$. Una ventaja importante de está unidad es que, a parte de integrar los sensores básicos para determinar el estado del helicóptero en el aire, se compró a una empresa que se dedica a la investigación en este tema, pero con código libre (licencia GNU), por lo

tanto, podremos alterar y modificar el código de la misma según las necesidades del proyecto.

1.2 Breves apuntes sobre los sensores empleados

En esta sección se explica el funcionamiento básico de los acelerómetros y los giróscopos.

1.2.1 Acelerómetros

Los acelerómetros utilizados en la IMU tienen salida de ancho de pulso modulado (PWM). El modelo empleado es el ADXL202 de la casa "Analog Devices Corp." con un rango de medida de ±2g y doble eje de medida.

Como los ADXL202 miden la aceleración en dos ejes perpendiculares y disponemos de dos de ellos es posible medir en las tres direcciones del espacio, quedando un 4 eje de redundancia o en nuestro caso inhabilitado.

La salida de un acelerómetro para uno de sus ejes tiene la forma mostrada en la Ilustración 2.

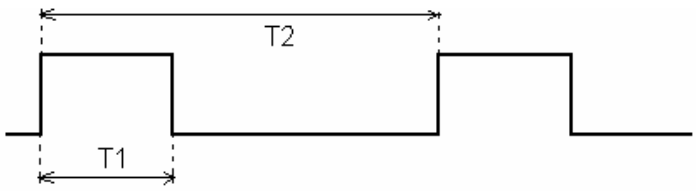


Ilustración 2- Salida PWM producida por un acelerómetro

Donde T2 es la duración completa de un pulso (período) y T1 es la parte activa del pulso. La relación entre ellas nos provee de una medida de la aceleración (en número de g´s) en un eje según la Ecuación 1.

$$A(g) = \frac{\frac{T1}{T2} - 0.5}{k}$$

Ecuación 1 – Medida de aceleración en g´s según los anchos de pulso

Donde "k" es un parámetro que depende de la masa del modelo. Así, su valor para los modelos ADXL202 y ADXL210 es 0.125g y 0.500g, respectivamente.

Desafortunadamente, la existencia de dos ejes de medida complica ligeramente la medida simultánea debido a que es el punto medio de los pulsos T1 el que coincide en vez del comienzo de los mismos. Como se puede observar en la Ilustración 3.

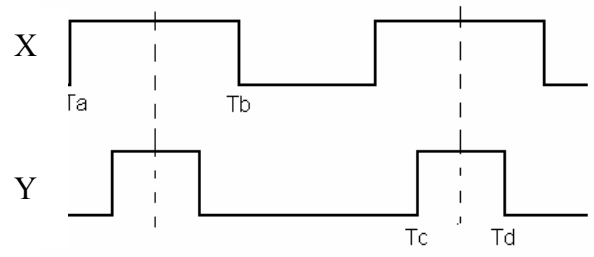


Ilustración 3

La manera de solucionar el problema es medir el pulso T1x = (T_b-T_a) primero y medir el pulso T1y = (T_d-T_c) después, utilizando el tiempo de

microprocesador entre ambas tareas para realizar cualquier operación necesaria.

1.2.2 Giróscopos

Un giróscopo es un sensor capaz de medir la aceleración angular en torno a un eje. Normalmente, están provistos de una entrada PWM utilizada como señal reloj para sus operaciones internas. En cuanto a las salidas, aparte de una señal por cada eje en el que midan la aceleración angular poseen una salida de temperatura interna y una referencia de voltaje "en reposo". Esto puede verse en la Ilustración 4.

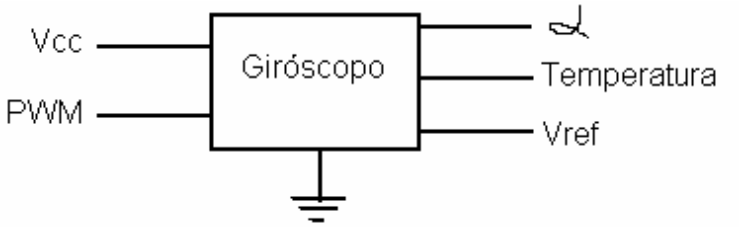


Ilustración 4 – Entradas y salidas de un giróscopo.

Existen dos clases ampliamente utilizadas en helicópteros RC (Radio Control): piezo-eléctricos y mecánicos. Los primeros suelen ser de bajo consumo y de pequeño tamaño y peso, pero la máxima aceleración angular que miden es baja y son bastante sensibles a la temperatura. En cambio, los mecánicos son más robustos ante cambios de temperatura, pero su consumo, peso y tamaño es mayor.

El proceso para obtener un valor del ángulo girado a partir de la medida del giróscopo pasa por una doble integración de la aceleración angular corregida con la temperatura.

Dado que las señales se presentan con ruido, para obtener una buena precisión se hace necesario el uso de métodos sofisticados de filtrado, por ejemplo, filtro de Kalman.

2 Hardware de la Unidad de Medida Inercial

2.1 Características principales

A continuación se muestra una descripción de la Unidad de Medida Inercial, el proceso de montaje necesario y un análisis a "grosso modo" de los diferentes componentes de la misma. La información ha sido extraída de [1].

Este sistema es una Unidad de Medida Inercial de seis grados de libertad en su revisión 2.4 diseñada por Rotomotion LLC, que mide la aceleración en los ejes X, Y, Z, y la rotación en torno a estos. Las señales analógicas provenientes de los sensores son amplificadas y filtradas convenientemente mediante un filtro pasa-baja, para reducir el impacto del ruido sobre las medidas resultantes.

Las aplicaciones más claras de este sistema son: el control de vuelo y la estimación de estado de un cuerpo. Además podría aplicarse en sistemas de realidad virtual.

Las características principales del mismo son:

- Tres giróscopos (filtrados y amplificados)
- Tres acelerómetros (filtrados y amplificados)

- 10 canales para control de servos normales.
- 1 canal para control de servos rápidos.
- Comunicación serie RS-232.
- Microprocesador ATMEL preprogramado. Bootloader para programación por vía serie.
- Puerto de programación tipo SPI adecuado para programadores Atmel STK-200 y STK-500.
- Biblioteca de decodificación de PPM para JR o Futaba.

La IMU consta de tres placas interconectadas. La primera o principal es la encargada del control de las otras dos y de los cálculos. Las otras dos placas son las sensibles a los distintos ejes del espacio, se tiene una que mide en los ejes XY y otra en el eje Z.

2.2 Descripción del hardware

La placa principal es la PLACA DE CONTROL. En la Ilustración 5 se puede ver la cara superior de la misma y en la Ilustración 6 la inferior:

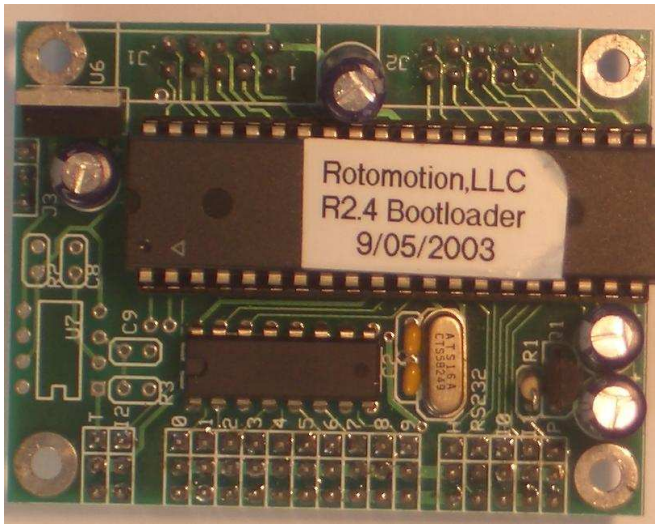


Ilustración 5 – Placa de Control. Cara superior

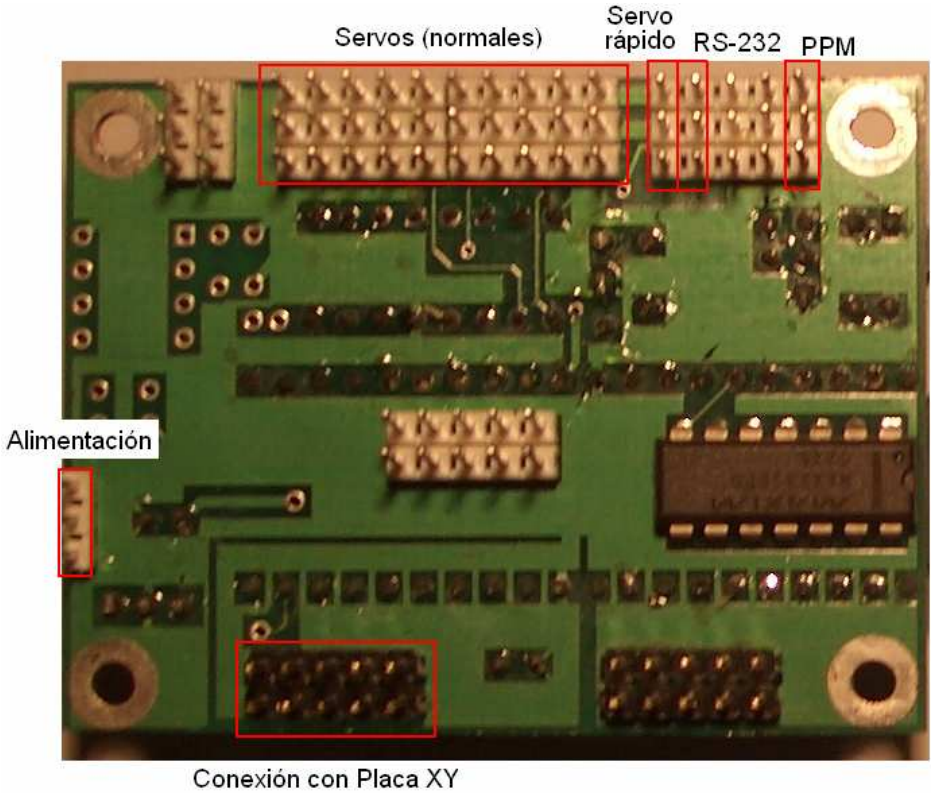


Ilustración 6– Placa de Control. Cara inferior

- Microprocesador ATMEL AtMega32, el cerebro de toda la unidad, encargado de capturar los valores que miden los sensores mediante los ADC –Analog to Digital Converter-, computarlos convenientemente (filtrado del ruido, filtro de Kalman,…), y dar las ordenes necesarias a los servos. También se ocupa de comunicar vía serie todas las medidas, órdenes, etc. Se debe decir que en el modo que se ha programado para este proyecto solamente se utilizan las siguientes funciones:
    - Captura de las medidas de los sensores mediante los ADC. Actualización cada 0,03276 segundos.
    - Captura de las señales PPM que provienen de la emisora.

- MAX-231: adaptador de tensiones para vía serie.
- Conector con Placa XY (marcado como J1).
- Conector de alimentación (marcado como J3).
- 10 conectores para servos normales (marcados del 0 al 9)
- Conector para servo rápido (marcado como H).
- Conector serie (marcado como RS-232).
- Conector para señal PPM (marcado como P).
- Cristal 8MHz
- Regulador de Tensión LM7805 con salida estable a 5V.
- Resistencias y Condensadores varios

La segunda placa es la PLACA XY. En la Ilustración 7 se pueden observar los componentes principales:

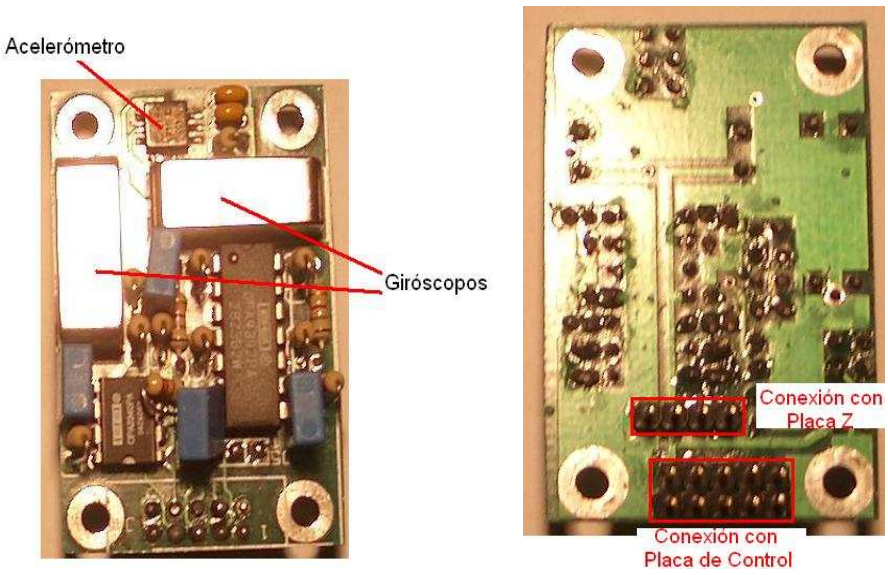


Ilustración 7 – Placa XY

- Acelerómetro ADXL202AE.

- Giróscopos CG16-DO.
- Amplificadores Operacionales Rail-to-Rail: OPA4342PA y OPA2340PA.
- Conector con Placa de Control (marcado como J4).
- Conector con Placa Z (marcado como J5).
- Resistencias y condensadores varios.

La tercera placa es la PLACA Z. En la Ilustración 8 se pueden observar los componentes principales:

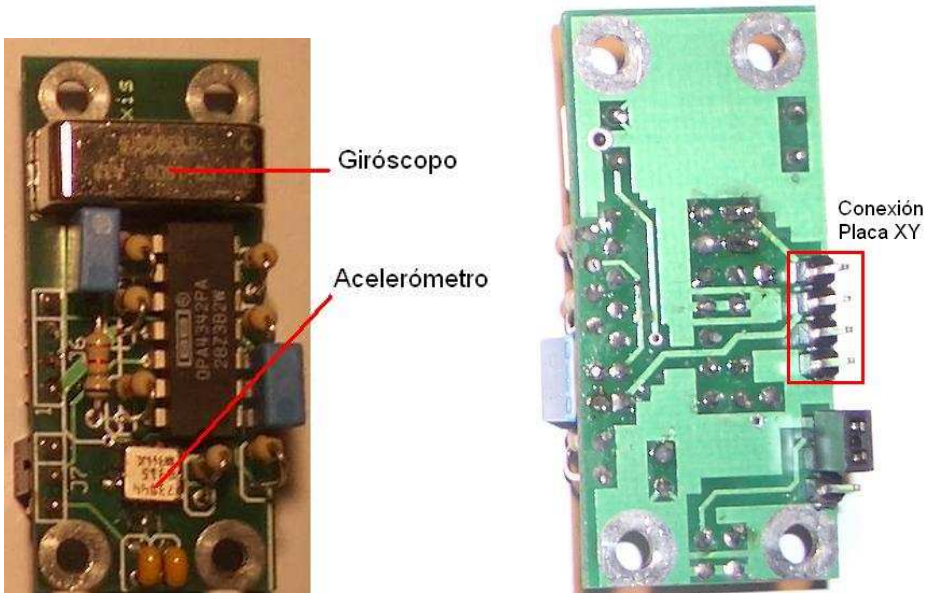


Ilustración 8 – Placa Z

- Acelerómetro ADXL202AE.
- Giróscopo CG16-DO.
- Amplificador Operacional: OPA4342PA.
- Conector con Placa XY (marcado como J6).
- Resistencias y condensadores varios.

2.3 Montaje y conexión

El montaje de la placa se realizó según la guía que se puede encontrar en [1].

La Ilustración 9 – Conexión entre las placas de la IMU es una imagen de las placas terminadas y de la conexión entre ellas.

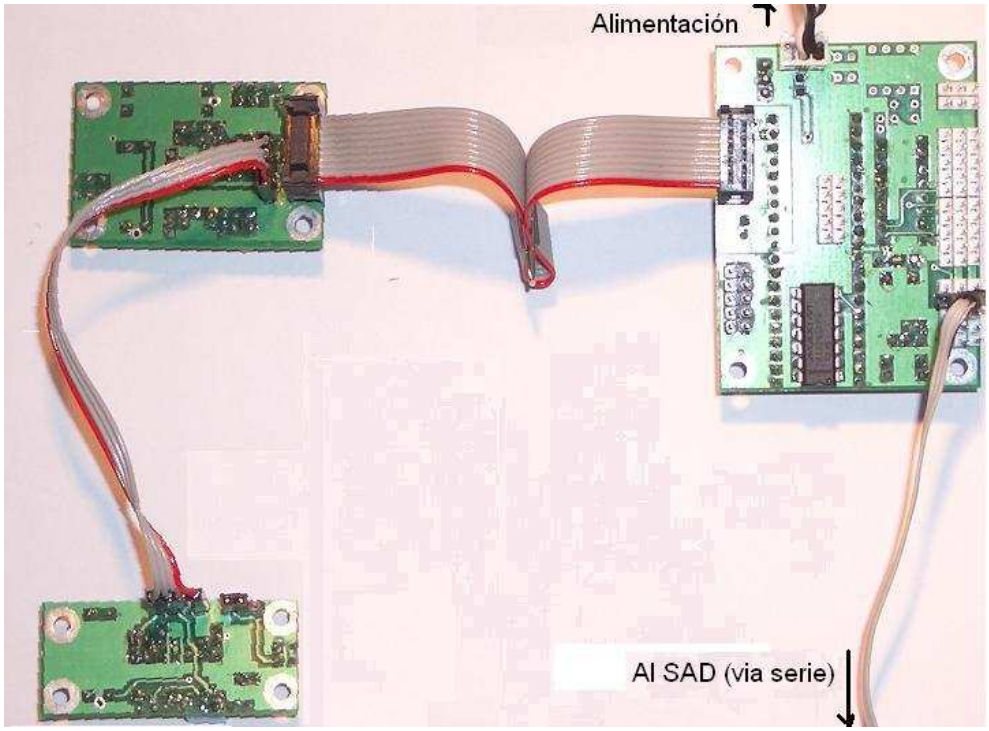


Ilustración 9 – Conexión entre las placas de la IMU

2.4 Modificación de los filtros anti-aliasing

Las pruebas realizadas en la segunda parte del proyecto realizada por Fernando Ángulo pusieron de manifiesto la existencia de una gran cantidad de ruido en las medidas realizadas con esta unidad en un helicóptero a escala como puede observarse en la Ilustración 10.

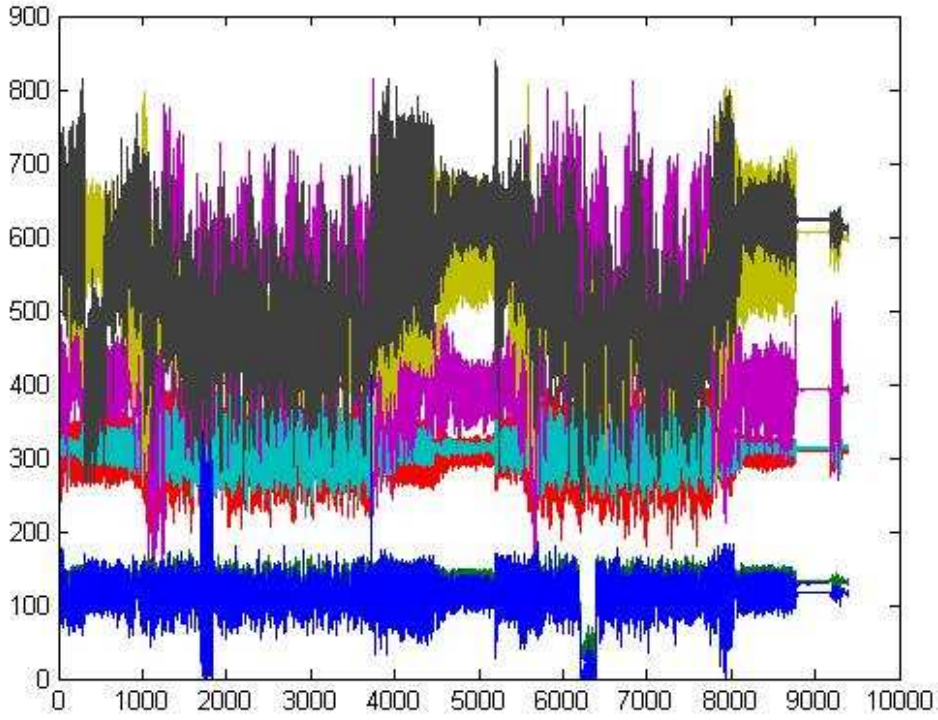
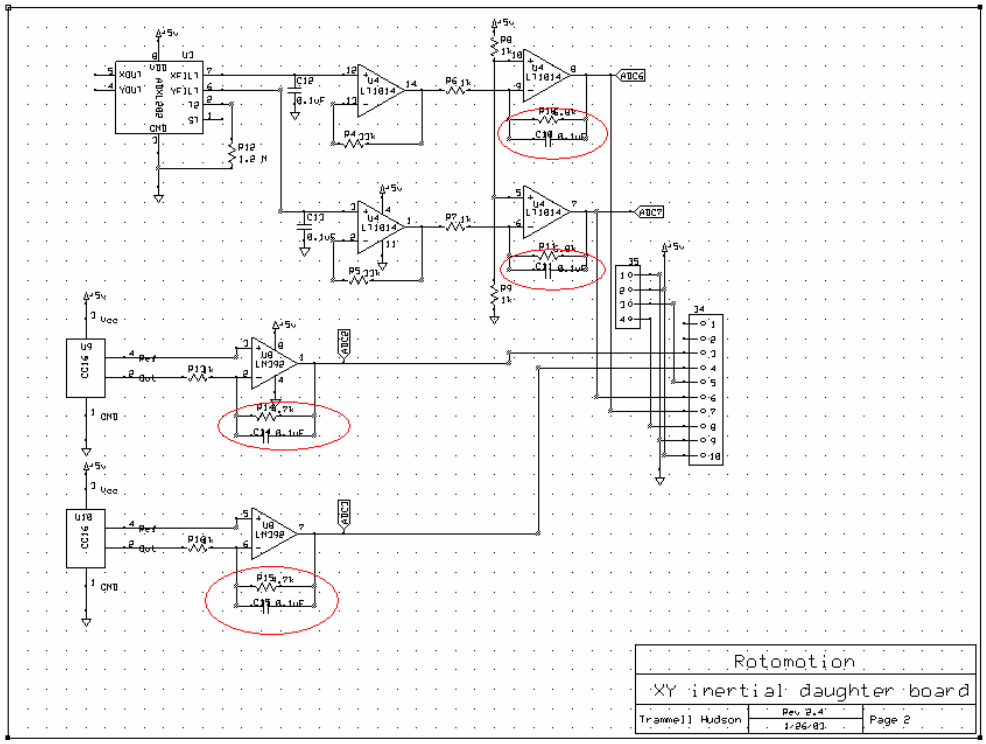


Ilustración 10 – Ruido en las medidas

Por la razón anterior se creyó conveniente la modificación de los filtros anti-aliasing del diseño original de Rotomotion para tratar de reducir el citado ruido. Dichos filtros son filtros analógicos pasa-baja, situados a la salida de los sensores (acelerómetros y giróscopos) y se muestran en la Ilustración 11.



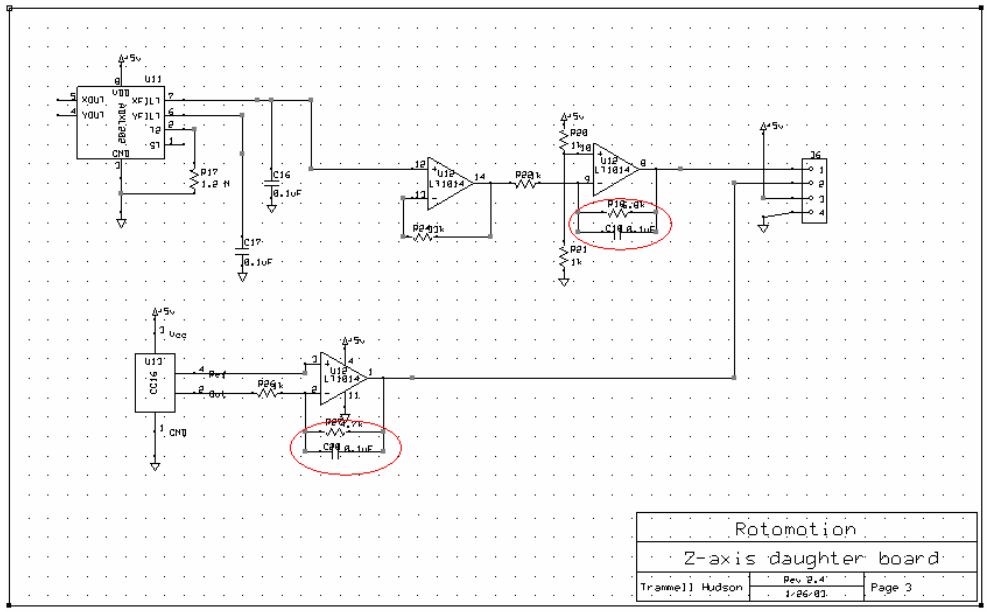


Ilustración 11 – Planos esquemáticos de las placas de medida de la IMU. Se han marcado en rojo los filtros antialiasing

La Ilustración 12 muestra los condensadores cambiados en cada una de las placas de medida para disminuir la frecuencia de corte de los filtros pasa-baja antialiasing. Los que están en amarillo son los originales de la

placa, los otros son de 220nF y 330nF, rojo y verde, respectivamente. Ninguna resistencia ha sido cambiada.

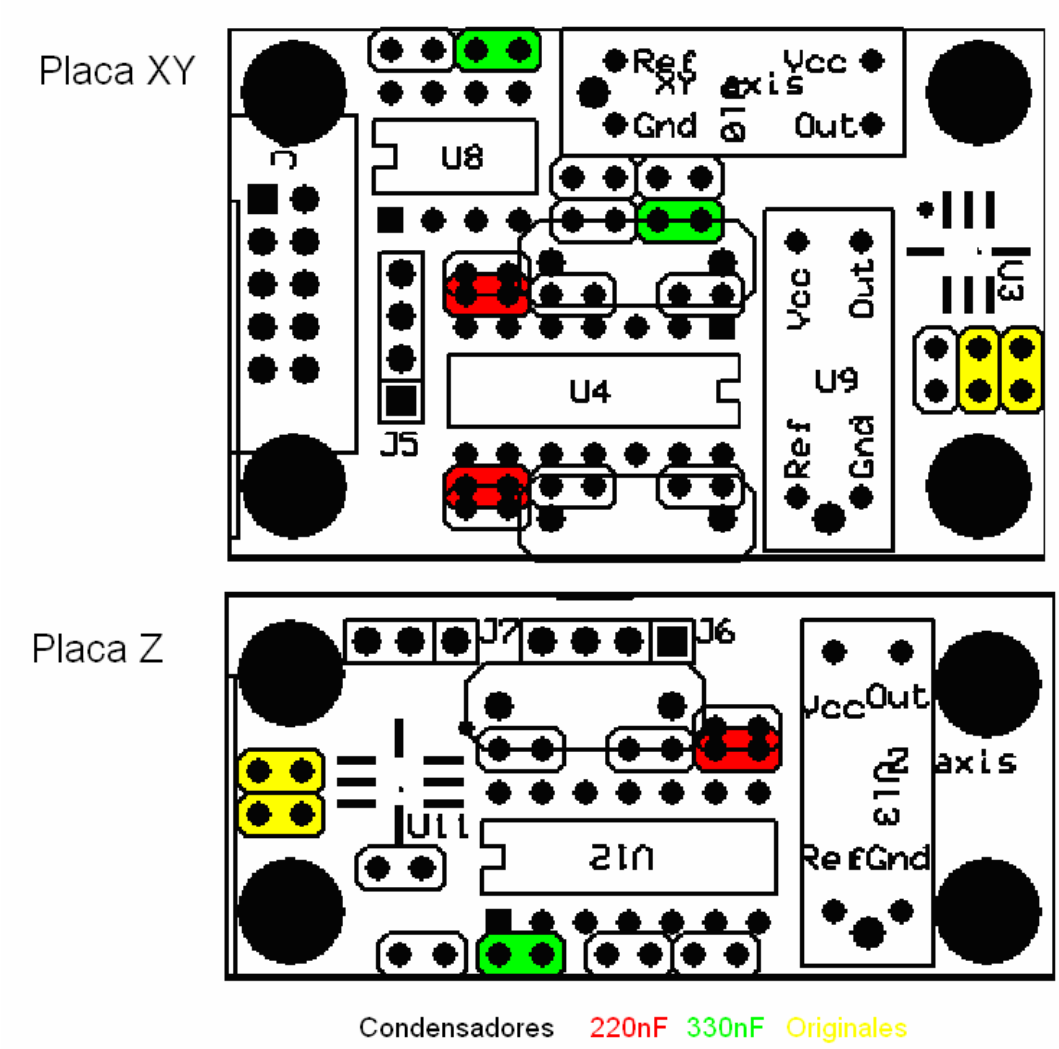


Ilustración 12 – Modificaciones realizadas en los filtros antialiasing

3 Software de la Unidad de Medida Inercial

3.1 Análisis

Como se ha comentado anteriormente la IMU tiene licencia GNU (libre). Aparejado a la gran ventaja que esto conlleva de facilidad de aprendizaje mediante el estudio directo de su hardware y software, lleva la

enorme desventaja de una documentación en muchos puntos deficiente o insuficiente. Por lo tanto, un paso previo a la utilización de esta unidad es lógicamente el estudio y compresión del software que maneja los recursos de la misma. Aprovechando este estudio se generará una documentación más detallada en los puntos de mayor importancia para este proyecto y una documentación básica que se podrá utilizar para las futuras ampliaciones del mismo. Toda la documentación utilizada para este apartado se encuentra en [3]**.** Además el código que se describe en la Tabla 1 se entrega en un CD al director del proyecto.

El siguiente cuadro muestra los archivos de código que se incluyen en la distribución de la IMU rev2.4, ordenados por categorías y acompañados de una breve explicación de su función. Las categorías en las que se ha clasificado son las cinco siguientes:

- Cálculos y control: Archivos encargados de alguna tarea de cálculo o incluso del control del helicóptero.
- Control de subsistemas de la IMU: Archivos para el control de los diferentes sensores adicionales que se pueden conectar a la IMU, como un tacómetro o una brújula.
- Control de periféricos del microprocesador y librerías de apoyo: Son archivos de código para el control de los periféricos del microprocesador optimizados para las necesidades del proyecto.
- Configuración: Archivos que contienen parte de la configuración general del sistema.

- Otros: Varios archivos con diferentes propósitos, por ejemplo, el bootloader y el lazo principal del sistema.

ARCHIVOS	DESCRIPCIÓN
CÁLCULOS Y CONTROL	
pcm.c	Decodifica la señal PCM del pin ICP.
Ahrs.h, ahrs.c y ahrs_data	Cálculo de posición y orientación del helicóptero. Versión para micros con memoria limitada. La actualización de los ADC se realiza cada 0,3276 segundos.
Pid.h y pid.c	Rutinas del control PID (Proporcional/Integral/Diferencial). Por ahora, solamente PD. Optimizado para microcontroladores con memoria limitada y sin unidad de coma flotante. Manda por vía serie sus resultados bajo la etiqueta GPAUT .
ppm.h	Cálculo de los anchos de las señales PPM y envío de los mismos a través del puerto serie.
Run_ahrs.c	Cálculo de orientación e inclinación del helicóptero para la rev 2.4 con la brújula tipo PNI de 3 ejes. Toda la potencia de cálculo y toda la memoria se gasta para el procesamiento (se lanzan por vía serie los resultados), así que no se pueden utilizar servos, etc.
Fast_ahrs.h y fast_ahrs.c	Cálculo de posición y orientación del helicóptero.

Servo.h y servo.c	Control de los servos mediante el contador de 16 bits. Se genera un ancho de pulso según el valor entre 0 y 65536. 0 es "todo a la izquierda" (pulso de 1.0ms) y 65536 es "todo a la derecha" (2.0ms).
Imu.h e imu.c	Código de manejo de la IMU. La función calib_init() debe llamarse antes que esto para poder leer los parámetros de los acelerómetros de la EEPROM.
CONTROL DE SUBSISTEMAS DE LA IMU	
V2x.h	Para la brújula en modo V2X.
pni.h	Para la brújula en modo PNI.
button.h	Lectura de botones en la rev2.2 (la nuestra es la rev2.4)
tach.h	Tacómetro.
CONTROL DE PERIFÉRICOS DEL MICROPROCESADOR Y LIBRERÍAS DE APOYO	
Adc.h y adc.c	Código para la conversión de valores analógicos de los sensores a digitales.
Mat.h y mat.c	Librería de tratamiento de matrices optimizada para microcontroladores con memoria limitada.
Soft_uart.h y soft_uart.c	
String.h y string.c	Funciones básicas para salida de cadenas y números.

Uart.h y uart.c	Control de USART (Universal Sincronous and Asincronous serial Receiver and Transmiter)
timer.h	Control de timers
CONFIGURACIÓN	
Calib.h y calib.c	Código para recalibrar y almacenar los valores de los acelerómetros en la EEPROM del Mega163.
config.h	Selección de modo PPM o PCM y configuración de los bancos de servos.
OTROS	
mainloop.c	Lazo principal. Actualiza los servos y manda por la vía serie los valores PPM (etiqueta: GPPPM), ADC (etiqueta: GPADC) y de la brújula (etiqueta: GPHDG).
loader.c	Bootloader de la IMU.
assembly.S	Rutinas de lectura/escritura en la flash y la EEPROM para el bootloader.
memory.h	Manipulación de espacios de memoria
average.h	Añade un valor a la ventana de media móvil
ins.c	Cálculo de posición y orientación en 3D (AHRS_3D)

Tilt.h y tilt.c	Cálculo de la inclinación usando un acelerómetro de dos ejes y un giróscopo de un eje, relacionados a través de un filtro de Kalman cuyos estados son el ángulo y los bias del giróscopo.

Tabla 1 – Archivos de la IMU rev2.4 en Autopilot.

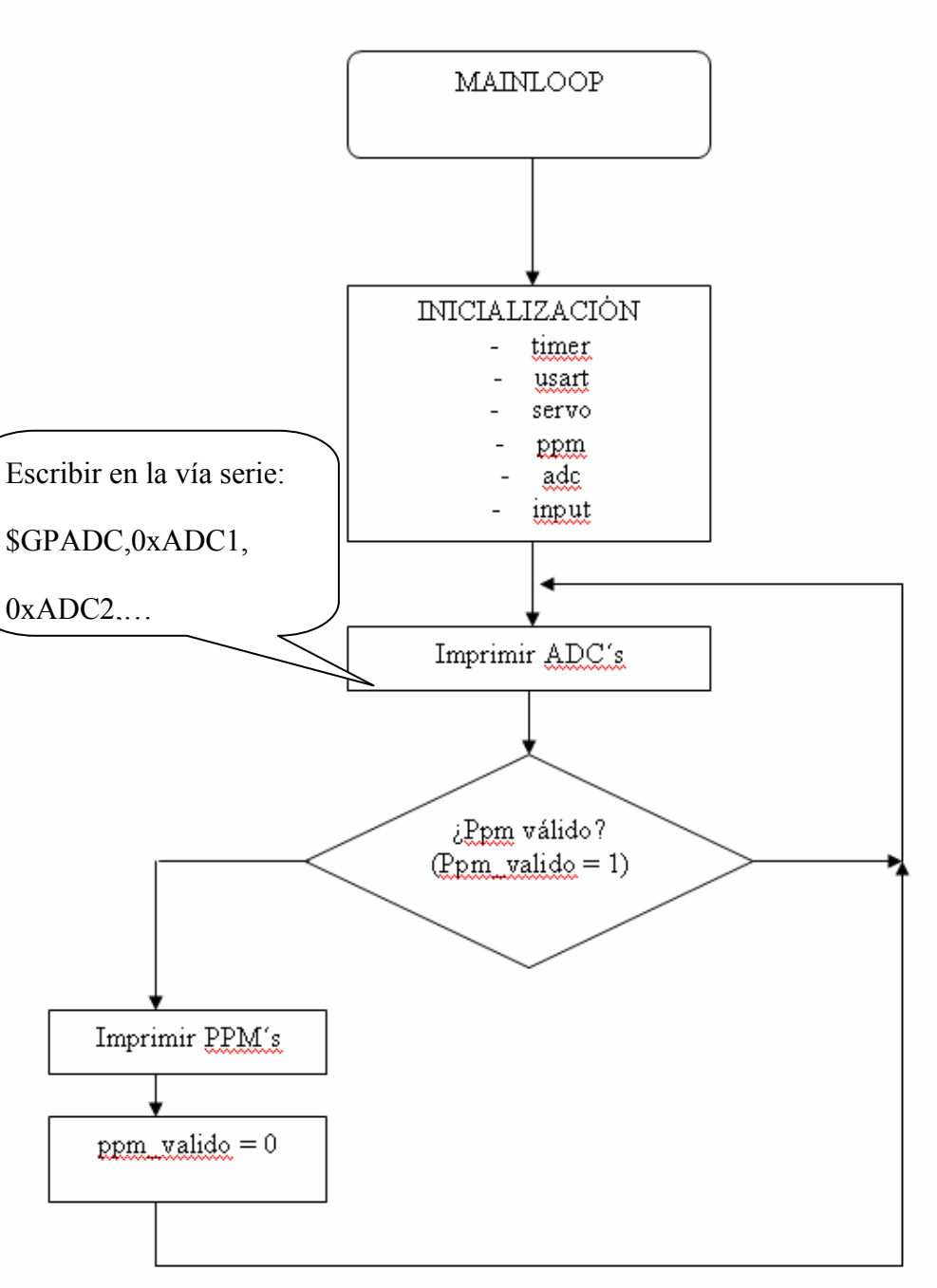
Como se puede apreciar por lo dicho hasta el momento en este apartado, la IMU adquirida en Rotomotion LLC tiene capacidad para calcular el estado del helicóptero y actuar sobre los servos para corregir las desviaciones que sucedan durante el vuelo mediante, como ya se explicará, un control PID. La pregunta que surge en este punto es por qué, si el propósito último de este proyecto es servir de infraestructura a otros proyectos que consigan esos objetivos, no se utiliza esta unidad directamente. Hay varias razones:

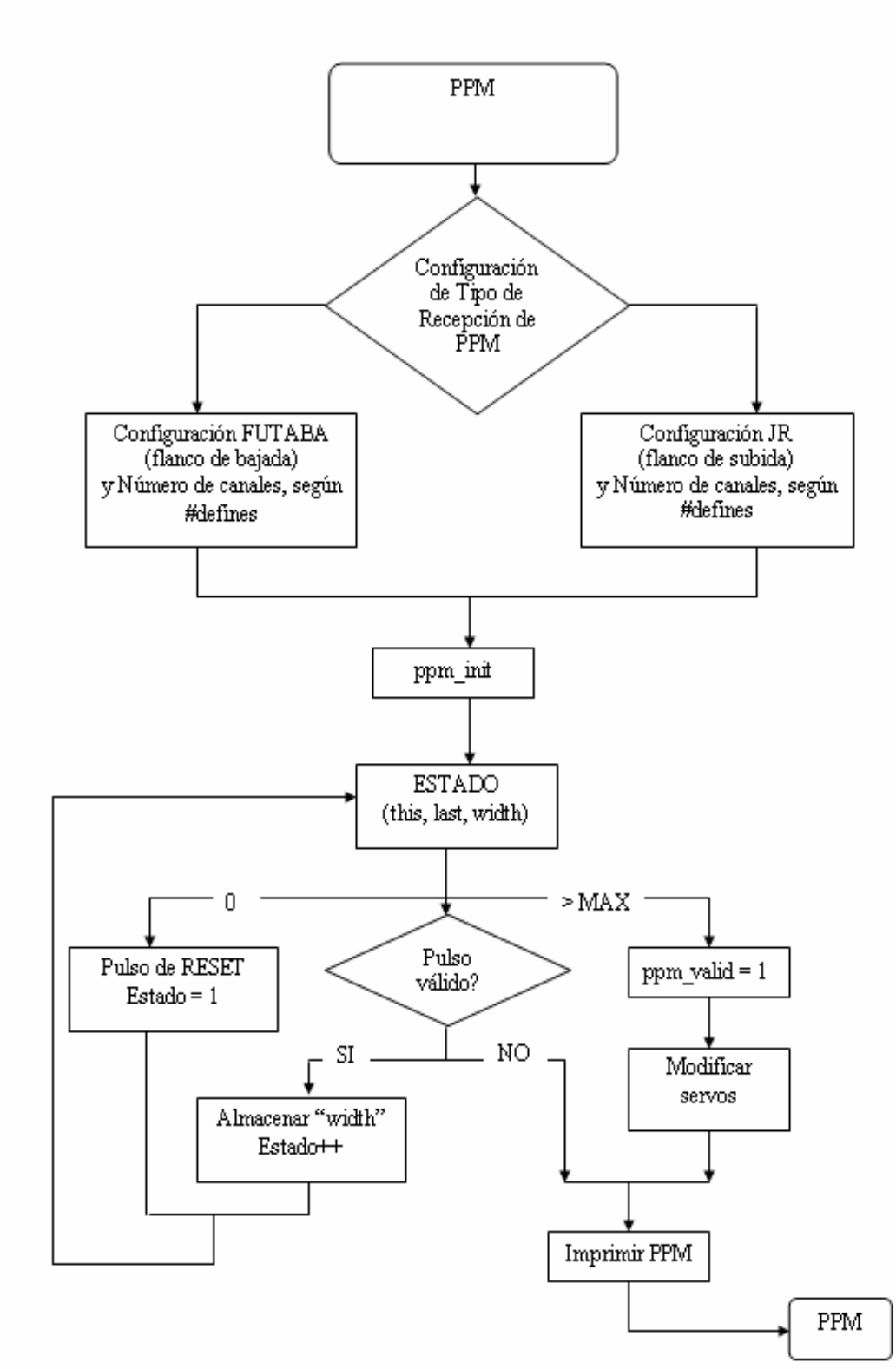
- Aunque la unidad tiene un control muy sencillo -sólo permite estabilizar en vuelo estacionario-, su análisis ayudará a una más rápida comprensión de cómo seguir en la investigación puesto que tiene licencia GNU, lo que permite un estudio exhaustivo.
- El control que usa esta unidad es un control PID muy sencillo que tiene en cuenta, lógicamente, el estado –posición y orientación- del helicóptero para corregir las desviaciones sufridas durante el vuelo, pero no tiene en cuenta su modelo

dinámico, que es necesario para poder hacer un control fino y preciso del helicóptero.

- El almacenamiento de los datos de los acelerómetros y los giróscopos (ADC), así como de las posiciones de los servos (PPM) en cada momento del vuelo, permitirá mediante un procesado exterior en un ordenador realizar estimaciones de estado y modelos dinámicos mucho más precisos y complejos que utilizando la limitada potencia de cálculo a bordo del helicóptero. A partir de esos modelos y estimaciones será posible en futuros proyectos realizar sistemas de control tremendamente más eficaces, seguros y precisos que el que actualmente se integra en la IMU.

A continuación se muestran los diagramas de flujo de los archivos de código que tienen una relación directa con la adquisición de datos:




PPM
Configuración de Tipo de Recepción de PPM
Configuración FUTABA (flanco de bajada) y Número de canales, según #defines
Configuración JR (flanco de subida) y Número de canales, según #defines
ppm_init
ESTADO (this, last, width)
0
> MAX
Pulso de RESET Estado = 1
Pulso válido?
ppm_valid = 1
SI
NO
Almacenar "width" Estado++
Modificar servos
Imprimir PPM
PPM

3.2 Modificaciones a nivel de software

Para reprogramar la IMU es necesario trabajar con sistema operativo Linux, porque los archivos de código de la misma están preparados con su "Makefile" para que así se haga.

Los archivos que se modifican para la configuración y adaptación del código son los que se explican a continuación. El código completo modificado se anexa en el "Anexo III" donde los cambios realizados se encuentran en color rojo para ayudar al lector.

3.2.1 Mainloop.c

Si una brújula está disponible en el sistema, al principio de "mainloop.c" se coloca la etiqueta *#define CONFIG_PNI* o *#define CONFIG_V2X* para elegir el modo de funcionamiento de la misma. Si, como en nuestro caso, no existe brújula alguna dichas etiquetas se deben omitir (*//#define CONFIG_PNI*).

En la función *main* se ha reordenado el código para hacerlo más efectivo.

3.2.2 Config.h

Como se explicó anteriormente este archivo es de configuración de bancos de servos y entrada PPM o PCM según la versión y las necesidades propias.

Las opciones que se dan son:

1. Elección de banco de servos normales entre SERVO_BANK_A y SERVO_BANK_B. Como en la versión de hardware rev2.4 no existe banco de servos B, hay que eliminar la etiqueta (*//#define SERVO_BANK_B*).

2. Selección de servo de alta velocidad SERVO_HS_A o SERVO_HS_B. Como no se utilizan servos de este tipo, se eliminan las dos etiquetas (*//#define SERVO_HS_A* y *//#define SERVO_HS_B*).

3. La decodificación PPM o PCM debe ser seleccionada para el correcto tratamiento de las señales provenientes desde la emisora. En este proyecto se utiliza PPM por su simplicidad manejo y modificación, así que se elimina la etiqueta *PCM_INPUT* (*//#define PCM_INPUT*) dejando activa #define PPM_INPUT.

3.2.3 Ppm.h

Se debe definir PPM_RX_TYPE como ‘1’ o como ‘2’ para establecer la recepción de señales desde la emisora como tipo Futaba o tipo JR, respectivamente. Futaba y JR son dos de los principales fabricantes de material electrónico para aeromodelismo del mundo (emisoras, giróscopos, servos, etc.). Cada uno de ellos tiene su propio estándar de señales en PPM. Realmente la temporización de las mismas es muy parecida, pero se diferencian en la forma de transportar la información; mientras la señal Futaba contiene la información en los anchos “altos” de pulso, JR la

contiene en los "bajos" (ver Ilustración 13 – Información en los PPM tipo Futaba y JR).

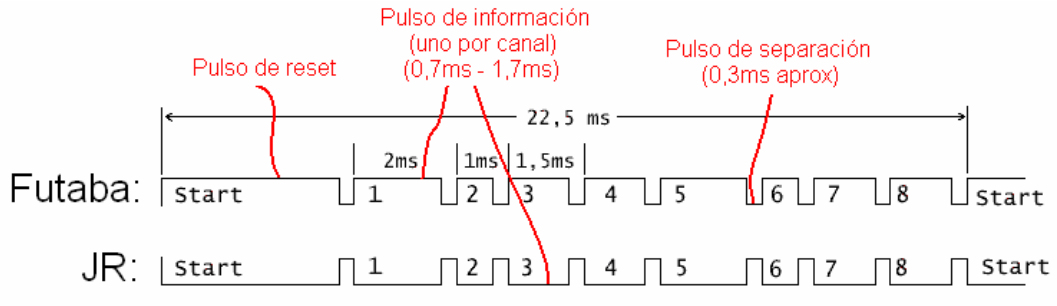


Ilustración 13 – Información en los PPM tipo Futaba y JR

Una vez establecido el tipo de recepción PPM, es necesario elegir el número de canales (PPM_MAX_PULSES) que tiene la emisora que se va ha utilizar que para este proyecto es 6.

3.2.4 Servo.c

El cambio que hay que realizar en este archivo es muy sencillo y tiene que ver con el cambio realizado en el anterior. Si en "ppm.h" se ha seleccionado *PPM_RX_TYPE = 1* (Futaba) deberá aparecer la línea *#define CONFIG_FUTABA*, mientras que si se ha elegido *PPM_RX_TYPE = 2* (JR) deberá aparecer la línea *#define CONFIG_JR*.

3.2.5 Makefile

Este archivo es un archivo de configuración de compilación del proyecto para trabajo en entorno Linux. En el se pueden seleccionar los archivos que se quieren incluir en la compilación para la creación del

"ejecutable", también se pueden seleccionar características de configuración según la distribución y versión de Linux empleada.

4 Resultados parciales obtenidos con la IMU

Para comprobar el correcto funcionamiento de la unidad una vez realizadas las modificaciones anteriores, tanto de software como de hardware, se ha utilizado un sencillo programa gratuito, IMU-Viewer, que se puede encontrar en [3]. Así, la Ilustración 14 – Comprobación de funcionamiento de la IMU. muestra los tres acelerómetros y los tres giróscopos en correcto funcionamiento. El movimiento se realizó con la mano.

Ilustración 14 – Comprobación de funcionamiento de la IMU.

5 Futuras ampliaciones del Sistema de Medida

En este apartado se dan una serie de ideas para futuras ampliaciones que futuros proyectos podrían desarrollar.

5.1 Otras variables que convendría medir

Además de la medida de las variables inerciales hay otras que es conveniente digitalizar, aunque no son necesarias para el correcto control del movimiento del aparato durante un vuelo. Estas son:

A. Variables del motor. Garantizan la seguridad del aparato. Por ejemplo, revoluciones por minuto del motor, revoluciones por minuto del rotor, nivel de combustible, temperatura de los gases de escape, temperatura del motor, etc.
B. Variables de posición global (Latitud, longitud y altitud). Informan de si el aparato está donde queremos que este. Pueden medirse con un GPS, pero será necesario incluir algún tipo de sensor de ultrasonidos para las medidas de altitud cuando el aparato esté volando a baja altura, debido a la escasa precisión de los GPS en esa dirección.
C. Variables del equipo instalado a bordo, por ejemplo el zoom en cámaras fotográficas y videocámaras o cantidad que queda de venenos en tareas de fumigación de campos de cultivos.

El muestreo de estos tres nuevos grupos de variables puede ser más lento que el de las variables inerciales porque su relevancia para el control y seguridad del helicóptero es menor.

Capítulo 3 SISTEMA DE ALMACENAMIENTO DE DATOS (SAD)

1 Introducción

Recordemos que uno de los principales objetivos de este proyecto es realizar un posterior análisis de los datos proporcionados por la Unidad Inercial de Medida (aceleraciones lineales y velocidades angulares) para, mediante un proceso informático, determinar el Estado del helicóptero -orientación espacial-. Por lo tanto, una vez puesta en funcionamiento la IMU nace la necesidad de realizar un almacenamiento de las medidas provistas por esta.

Aunque el control del helicóptero no es objetivo de este proyecto, necesita, a parte de las medidas inerciales, la posición de los servos -que controlan el gas del motor y la inclinación de las palas del rotor- para que sea posible la identificación de parámetros del modelo dinámico. Por tanto, para facilitar futuras investigaciones se almacenarán también mediante este sistema las señales de los servos.

Para afrontar esta tarea es necesario el diseño e implantación de un sistema electrónico consistente esencialmente en algún tipo de memoria y en un microprocesador que controle la comunicación entre dicha memoria y la IMU.

La memoria seleccionada es del tipo SD por su gran capacidad de almacenamiento, sus facilidades de ampliación y su reducido peso y tamaño, hechos que se consideran una ventaja vital en futuras aplicaciones.

Tras la elección del tipo de memoria descubrimos una nueva e inesperada ventaja que nos conduce a tomar el microprocesador ATMega-32L de la casa ATMEL como cerebro de la tarjeta SAD puesto que la empresa "Progressive Resources, LLC" vende vía Internet una librería llamada FlashFile para el manejo de tarjetas SD a través de cualquiera de los microprocesadores de la gama ATMega (la elección el modelo 32L se explicará en el apartado 2.1 de este documento: "Etapa 1 – De la idea a los componentes").

2 Etapas de desarrollo

En este apartado se va a explicar el proceso de desarrollo que se ha seguido para la consecución del SAD, partiendo de la simple necesidad de almacenamiento de datos y acabando con la materialización de la idea en una placa perfectamente operativa, pasando por la solución adoptada para cada uno de los problemas que han ido surgiendo a lo largo del camino.

Se ha realizado una división en etapas con la intención de facilitar la toma de una idea general sobre el desarrollo del SAD. Estas etapas no son totalmente cronológicas, pero la estructura que se les ha dado facilita la comprensión de las mismas.

Las etapas de desarrollo que tienen que ver con la programación de la IMU y su funcionamiento se han explicado en "Fase I: Unidad de Medida

Inercial (IMU)" de esta memoria por considerar que ese era el contexto apropiado para un adecuado entendimiento de dichas etapas.

2.1 Etapa 1 - De la idea hasta los componentes.

Antes de empezar es conveniente indicar que los helicópteros a escala con motor de explosión son sistemas con gran vibración, por tanto, durante todo el diseño se tendrá muy en cuenta esté hecho para, en la medida de lo posible, independizar el sistema de estos problemas. Para ello se utilizarán medidas tales como condensadores de desacoplo o puntos de soldadura fuertes y realizados con extremo cuidado.

En la Ilustración 15 se esquematiza el flujo de información entre la Unidad Inercial y la memoria tipo SD mediante el ATMega. Los datos debidamente empaquetados parten por vía serie desde la IMU hacia el microprocesador el cual por medio de la vía SPI los guarda en la memoria SD.

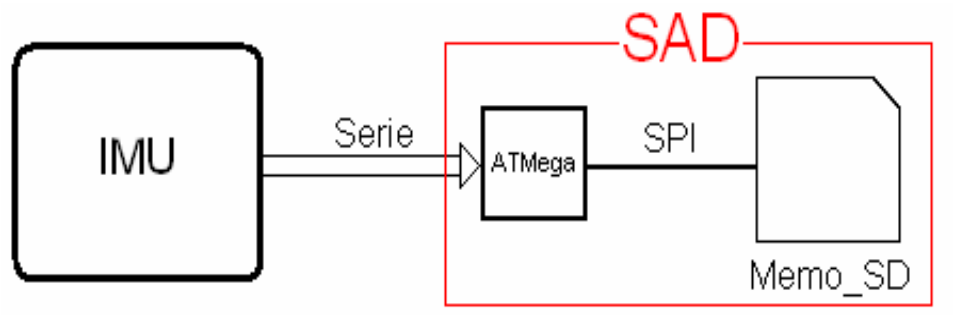


Ilustración 15 – Esquema básico de comunicación IMU-SAD

De los posibles micros dentro de la familia ATMega se escogió el ATMega-32L. El "32" indica la capacidad en kilobytes de almacenamiento

en memoria no-volátil (tipo flash), es decir, aquella que sirve para el guardado de programas. La letra "L" significa "Low Power Consumption" (Bajo consumo de energía). Las razones de dicha elección son dos: la primera, que se consideró adecuada la relación capacidad/precio para albergar la librería de manejo de la SD y la segunda que, debido a la escasa capacidad de carga que suele caracterizar a los helicópteros a escala, todo ahorro de energía supone un menor peso de baterías cuando el sistema se cargue en uno de estos.

La librería (a la que se hace referencia continuamente) utiliza el puerto PB7...4 en su función de vía SPI ("Serial Peripheral Interface", -Interfaz Periférico tipo Serie-) que es un puerto del tipo serie síncrono como modo de direccionamiento entre el microprocesador y la memoria. La utilización de este recurso se expondrá con más detalle en el Anexo.

La comunicación de la Unidad Inercial con el micro se realiza por vía serie asíncrona gracias al recurso USART ("Universal Synchronous and Asynchronous serial Receiver and Transmitter", -Receptor o Transmisor Universal del tipo Serie Síncrono o Asíncrono-) del ATMega que también se explicará detalladamente en el apartado 4.4 del Anexo I: Guía básica sobre manejo de AVR ATMEL Atmega.

Debido a que el ATMega viene de fábrica "limpio de programación" fue necesario dejar una comunicación auxiliar vía SPI para poder cargarle el "bootloader" (pequeño programa que da la capacidad de actualizar el código de programa del microprocesador mediante un mecanismo de auto-

programación escribe-mientras-lee). En el apartado 4.4 se desarrollará lo referente al “bootloader”.

Para garantizar un suministro estable de tensión de 3.3V que es la de trabajo del microprocesador se coloca un Regulador de Tensión modelo LD33V de “ST Microelectronics”.

Puesto que la tensión de trabajo del micro es 3.3V y el nivel típico de toda vía serie asíncrona como la empleada en la comunicación de este con la IMU o con un ordenador personal es de 5V, ha sido necesario colocar un adaptador de niveles lógicos para dichas tareas. El elegido es el MAX-3223.

Otros componentes necesarios para el funcionamiento del sistema son resistencias, condensadores, diodos LED y conectores varios. La mayor parte de los condensadores utilizados son para el desacoplo entre etapas de tensión. Hay otros necesarios para el propio funcionamiento del MAX-3223 y del micro -por ejemplo, para el cristal de 8MHz que utiliza el micro para crear la señal de reloj-. Uno de los condensadores y una de las resistencias han sido empleadas para crear una red RC en la patita RESET del micro. Los cuatro conectores son las uniones del SAD con el mundo exterior: zócalo de la memoria SD, intercambio de datos vía SPI, vía auxiliar para la carga del Bootloader, intercambio de datos vía serie asíncrona y alimentación de tensión. Los diodos LED servirán para visualizar fácilmente el estado del sistema creado.

A continuación se incluye un listado funcional a modo de resumen de todos los componentes que forman el Sistema de Adquisición de Datos (Tabla 2 – Componentes del SAD):

	Componente	Función	Comentarios
1	ATMega-32L	Cerebro del SAD: controla el flujo de información entre la memoria SD y la IMU.	Se comunica con la SD por medio del SPI y con la IMU por vía serie RS-232.
1	Conector_SD	Zócalo donde colocar la memoria SD para unirla eléctricamente con el microprocesador.	Facilita el trabajo con la memoria.
1	MAX-3223	Adaptación de niveles lógicos	
1	LD33V	Regulador de tensión	Salida a 3.3V
1	Conector x 6	Conexión auxiliar para la carga del "bootloader"	Conectado a la vía SPI
1	Conector x 3	Conexión de las baterías al regulador de tensión	Dos posibles conexiones
1	Conector x 3	Conexión de la IMU al adaptador de niveles lógicos	También es la posible conexión del micro a un ordenador personal
2	Diodo LED	Indicadores del estado del programa	Las diferentes combinaciones de colores se contarán en el apartado "3. Manual de Uso"
2	Jumpers	Uno para programación y otro para manejo de la alimentación.	
1	Cristal 8MHz	Señal de reloj del micro	

13	Condensadores		
	C1 (1μF)		
	C2 (1μF)		
	C3 (1μF)	Desacoplo	
	C4 (1μF)	Desacoplo	
	C5 (1Mf)	Desacoplo	
	C6 (56pF)	Red del Cristal	
	C7 (56pF)	Red del Cristal	
	C8 (100nF)		
	C9 (100nF)		
	C10 (1μF)	Red RC para RESET	
	C11 (1μF)	Desacoplo	
	C12 (1μF)	Desacoplo	
	C13 (1μF)	Desacoplo	
2	Resistencias		
	R1 (0kΩ)		
	R2 (10kΩ)	Red RC para RESET	

Tabla 2 – Componentes del SAD

2.2 Etapa 2 - Desarrollo con Eagle®.

El siguiente paso, antes de la construcción de la placa, es la realización de los esquemas en con algún tipo de programa de diseño. El programa elegido para tales efectos es Eagle®.

2.2.1 Introducción de componentes.

Eagle® recibe del usuario un esquema en el que se especifican los componentes que componen la placa, en este esquema los componentes son "cajas negras" de las que el programa solo conoce sus características constructivas ("pineado", tamaño, forma,…) y una serie de limitaciones de conexión (por ejemplo, los pines que tienen que estar obligatoriamente conectados). Normalmente, los componentes se encuentran ya caracterizados dentro de las propias librerías de Eagle®, pero en ocasiones el usuario tiene que realizar dicha tarea, mediante el propio editor del programa. Este es el caso del módulo **"**SDconnector**"** que fue creado especialmente para este proyecto a partir de su hoja de características.

2.2.2 El conector para tarjetas tipo SD.

La Ilustración 16 – Diagramas dimensional y funcional del zócalo para tarjetas SD (Las dimensiones reales se muestran en el Anexo IV: Características del conector/zócalo para tarjetas SD) muestra la libreria (.lbr) "SDconnector" tal y como se creó con el editor de Eagle® a partir de las

características dimensionales y funcionales del conector para SD extraídas de la hoja de características que se encuentra en el "Anexo IV: Características del conector/zócalo para tarjetas SD".

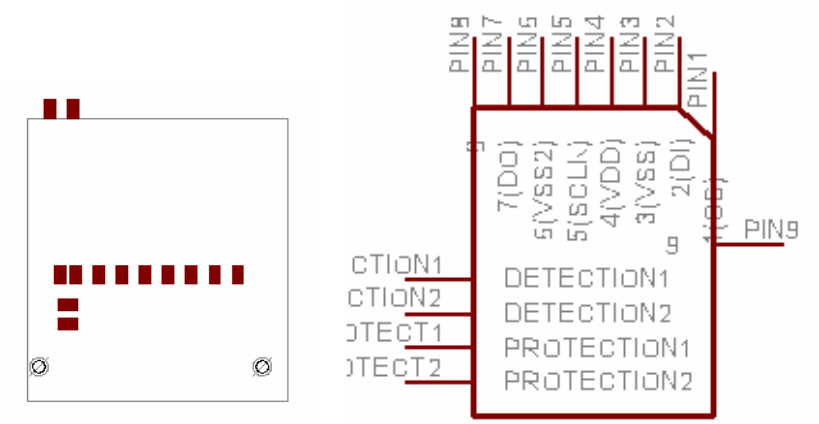


Ilustración 16 – Diagramas dimensional y funcional del zócalo para tarjetas SD (Las dimensiones reales se muestran en el Anexo IV: Características del conector/zócalo para tarjetas SD)

Los contactos designados como PIN1 a PIN9 son los pines de información estándar de cualquier tarjeta de memoria tipo SD, cuyas funciones se explicarán más adelante. Los contactos designados como DETECTIÓN1 Y DETECTION2 son pines para la detección de la tarjeta SD en el zócalo y los designados como PROTECTION1 y PROTECTION2 son pines que proporcionar información sobre la protección contra escritura de la tarjeta. Para más información consultar el Anexo III.

2.2.3 Conexionado eléctrico.

Una vez terminada la introducción de todos los componentes en el esquema se debe realizar la conexión eléctrica necesaria para que el sistema

funcione como se desea. Queda así finalizado el Esquema de la placa. La Ilustración 17 – Esquema de conexionado muestra dicho esquema (.sch):

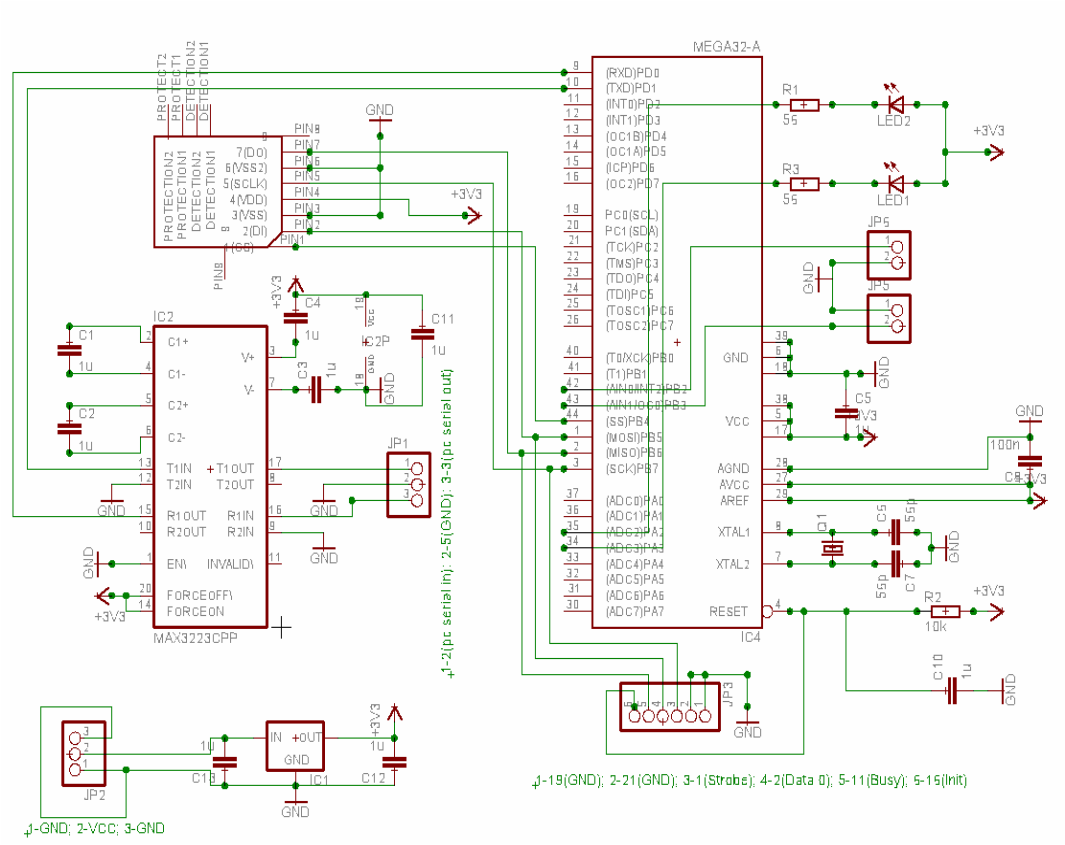


Ilustración 17 – Esquema de conexionado

2.2.4 Disposición real de componentes.

Con el esquema de la Ilustración 17 – Esquema de conexionado el programa pasa a una segunda pantalla de desarrollo en la que los componentes aparecen con su forma verdadera. Se le pide al usuario su ubicación -que debe ser realizada de la forma más óptima posible- para que posteriormente el programa calcule automáticamente los caminos que debe tener cada una de las pistas, que fueron establecidas en la pantalla anterior. El usuario elige el número de capas que quiere para su placa –en nuestro

caso una capa a dos caras- y tiene la opción de colocar todas las pistas manualmente o de que el programa genere las pistas y luego modificar alguna de estas pistas. Una vez que las pistas están colocadas a gusto del usuario este Esquema Real puede imprimirse (en papel para trasparencias) mostrando solo las capas de pistas para su posterior isolación. En la Ilustración 18 se muestra dicho diseño de pistas (.brd):

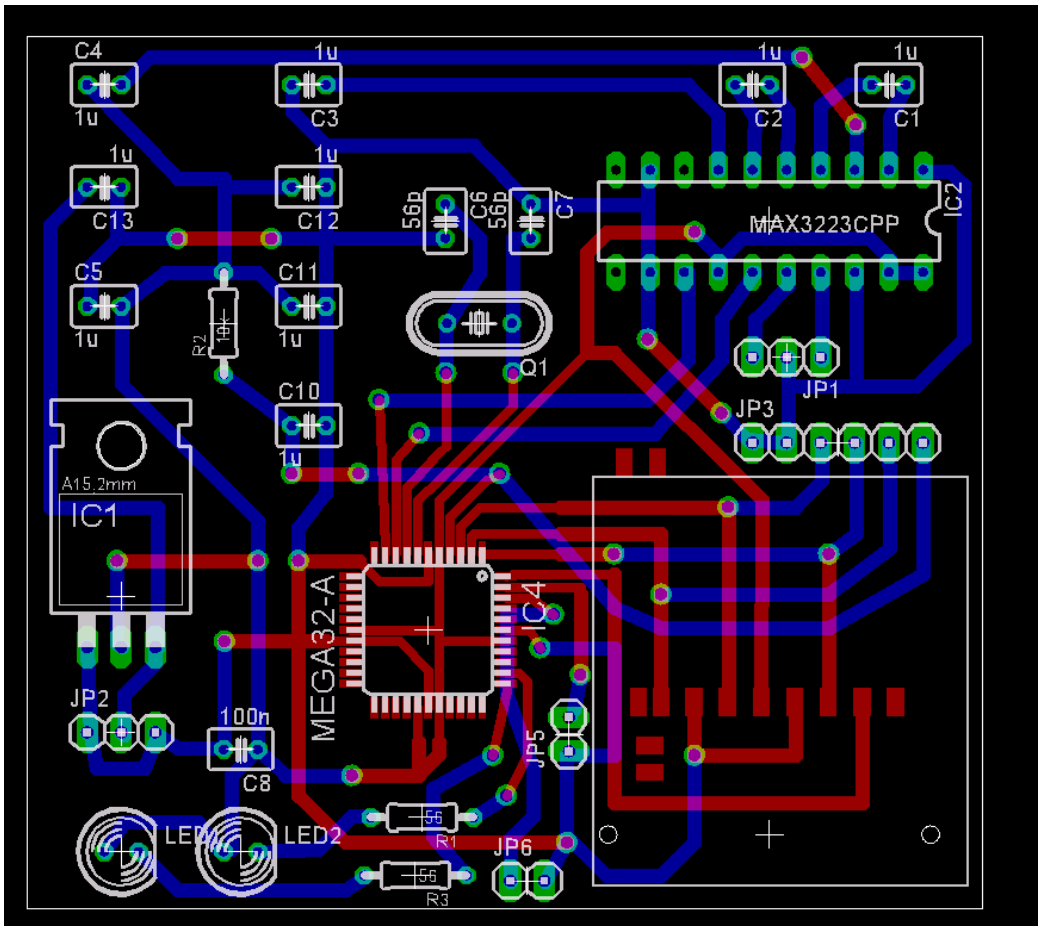


Ilustración 18 – Diseño de pistas del SAD

La correcta colocación de los componentes pasó por analizar la función de cada componente. Así el microprocesador, cerebro de la placa, está situado en el centro de la misma porque es uno de los componentes con más

conexiones y hace de vínculo entre los demás componentes. El zócalo para la SD está situado en un lateral y con la abertura orientada hacia fuera por facilidad de extracción. El MAX-3223 está colocado en un lateral y en paralelo al micro. Y el resto de componentes (condensadores, regulador de tensión, resistencias,…) están colocados para minimizar los cortes entre pistas. En la Ilustración 19 – Sistema de Almacenamiento de Datos se puede observar el aspecto final de la placa fabricada.

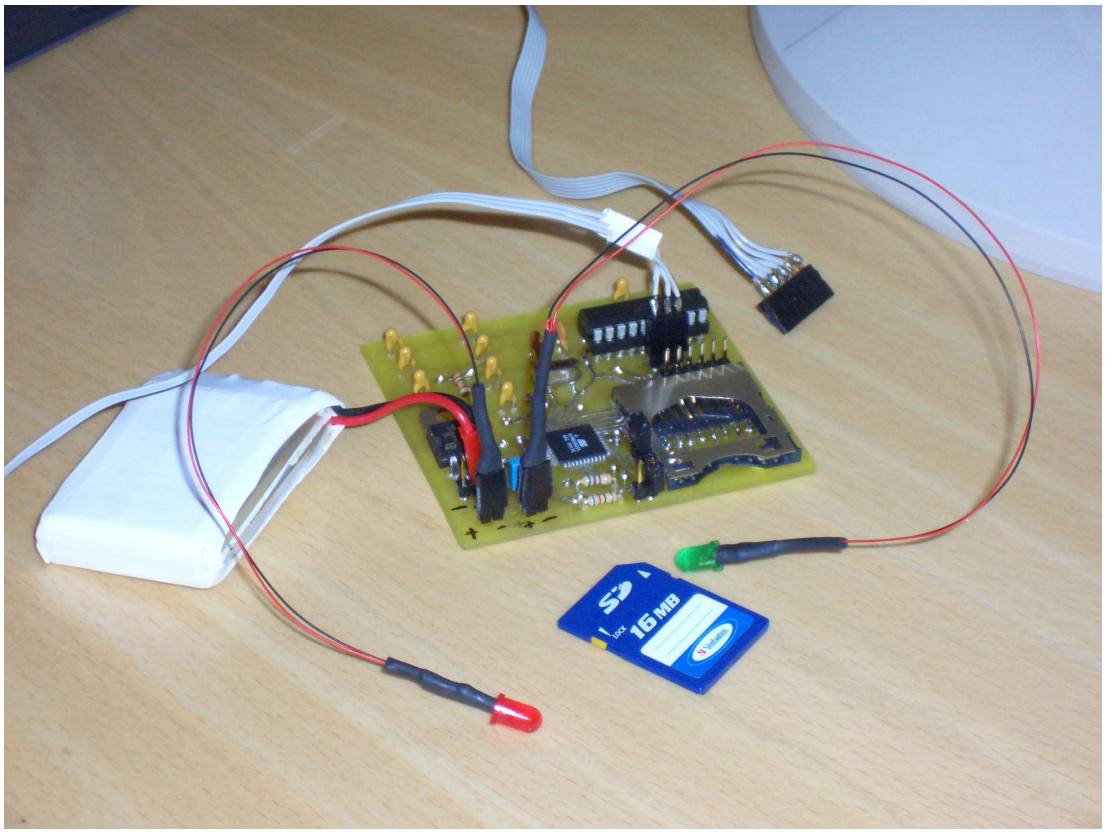


Ilustración 19 – Sistema de Almacenamiento de Datos

2.2.5 Tablas y resúmenes.

La Tabla 3 muestra el "pineado" de la placa:

Nº Pin	Nombre	Función
ATMega 32-L		
9	PD0 (RXD)	Función especial del puerto D como receptor de datos vía serie
10	PD1 (TXD)	Función especial del puerto D como transmisor de datos vía serie
44	PB4 (SS)	Función especial del puerto B como Chip-Select del SPI
1	PB5 (MOSI)	Función especial del puerto B como MOSI –Master Out Slave IN- del SPI
2	PB6 (MISO)	Función especial del puerto B como MISO –Master In Slave Out- del SPI
3	PB7 (SCK)	Función especial del puerto B como señal de reloj del SPI
39,6,18,28	GND	Puesta a tierra
38,5,17	VCC	Tensión de alimentación (3.3V)
27	AVCC	Tensión de alimentación del puerto A y del conversor A/D
29	AREF	Referencia analógica del conversor A/D
7	XTAL2	Salida del amplificador para el oscilador
8	XTAL1	Entrada del circuito de reloj
4	RESET/	Entrada de reset exterior

MAX-3223		
1	EN/	Entrada de habilitación (enable). Siempre encendido (GND)
2,4	C1+, C1-	Condensador de trabajo (1 μF)
3,7	V+,V-	Alimentación y referencia
5,6	C2+,C2-	Condensador de trabajo (1 μF)
13,17	T1IN, T1OUT	Los datos que han sido mandados por el micro se reciben en T1IN, el MAX-3223 adapta los niveles lógicos a los de la vía serie tradicional y salen por T1OUT hacia el receptor
16,15	R1OUT, R1IN	Los datos que han sido mandados por vía serie se reciben en R1IN, el MAX-3223 adapta los niveles lógicos a 3.3V y salen por R1OUT hacia el receptor del micro
9	R2IN	No se utiliza, pero para que no haya problemas de pines flotantes se ha colocado a GND
12	T2IN	No se utiliza, pero para que no haya problemas de pines flotantes se ha colocado a GND
14,20	FORCEON, FORCEOFF/	Siempre encendido (VCC)

REGULADOR TENSIÓN		
1	VI	Entrada desde batería
2	GND	Tierra, referencia
3	VO	Salida regulada
SD-CONNECTOR		
1	CS	Chip-Select
2	DI	Entrada de datos
5	SCLK	Reloj de sincronización
7	DO	Salida de datos

Tabla 3 – Resumen del layout del SAD

2.3 Etapa 3 – Fuse Bits y Bootloader.

Los Fuse Bits de un microprocesador son unos bits para la configuración de aspectos como la velocidad de arranque, la memoria reservada para el bootloader, la frecuencia del cristal, etc…

Las dos imágenes siguientes han sido extraídas del manual del ATMEGA32-L y muestran los diferentes Fuse Bits.

Table 105. Fuse High Byte

Fuse High Byte	Bit No.	Description	Default Value
OCDEN[4]	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN[5]	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN[1]	5	Enable SPI Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT[2]	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 100 for details)	0 (programmed)[3]
BOOTSZ0	1	Select Boot Size (see Table 100 for details)	0 (programmed)[3]
BOOTRST	0	Select reset vector	1 (unprogrammed)

Table 106. Fuse Low Byte

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out Detector trigger level	1 (unprogrammed)
BODEN	6	Brown-out Detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)[1]
SUT0	4	Select start-up time	0 (programmed)[1]
CKSEL3	3	Select Clock source	0 (programmed)[2]
CKSEL2	2	Select Clock source	0 (programmed)[2]
CKSEL1	1	Select Clock source	0 (programmed)[2]
CKSEL0	0	Select Clock source	1 (unprogrammed)[2]

Ilustración 20 – Fuse Bits del Atmega32L

La Tabla 4 muestra la configuración de los Fuse Bits para este caso concreto:

GRUPO DE BITS	VALOR	FUNCIÓN
CKSEL 3-0	1,1,1,1	Cristal Externo de 8MHz
SUT 1-0	1,0	Tiempo de arranque
CKOPT	1	Opciones del oscilador
BOOTSZ 1-0	1,1	Tamaño del bootloader
BOOTRST	0	Vector de reset
Resto	Predefinido	

Tabla 4 – Configuración de los Fuse Bits

Así obtenemos:

- High Byte: 0x9E
- Low Byte: 0xEF

Para la programación de estos bits se utiliza AVRDUDE.EXE que se explicará en el apartado 4.2 AVR Dude de Win AVR.

Un Bootloader es un código opcional que corre inmediatamente después del reset y que funciona como una especie de “mini-Sistema Operativo” dentro del microprocesador, permitiendo al usuario una más fácil

interacción con el micro en temas de programación, especialmente. Suele estar programado en lenguaje Ensamblador.

Para este proyecto se ha utilizado un bootloader que se explica en el apartado de recursos 4.4 Kargador y bootloader.

2.4 Etapa 4 – Adaptación de la librería a las necesidades del proyecto.

Lo primero es explicar las generalidades de manejo de la librería FlashFile de Progressive Resources LLC.

2.4.1 Descripción

FlashFile es un sistema de archivos para tarjetas de memoria Secure Digital (SD) o Multimedia Cards (MMC) que permite leer y escribir datos en formato FAT12/FAT16 usando microprocesadores del tipo Atmel AVR.

Una importante ventaja de trabajar en formato FAT12/FAT16 es que todo dato guardado en la tarjeta de memoria puede ser manejado directamente por un PC utilizando el lector correspondiente.

El conjunto total de funciones de dicha librería ocupa aproximadamente 20kB de la memoria programable tipo Flash del microprocesador cuando se trabaja con los estándares CodeVisionAVR y aproximadamente 30kB si se trabaja con ImageCraftAVR. Para este proyecto se ha decidido trabajar con CodeVisionAVR porque el microprocesador elegido (ATMega32-L) iría muy ajustado trabajando con ImageCraftAVR.

2.4.2 Consideraciones de memoria

Puesto que las tarjetas SD deben ser leídas o escritas en grupos de 512B es necesario un buffer de memoria SRAM en el microprocesador de este tamaño para tales acciones. Además es necesario establecer un tamaño del *heap* de 512B para el buffer, 40B para la información sobre la estructura del fichero y 4B más para el puntero a la base del fichero, es decir, 556B en total para cada fichero que deba estar abierto simultáneamente. En la aplicación que se ha dado a esta librería para este proyecto solamente se trata un fichero cada vez, por lo tanto, se ha configurado un tamaño de *heap* de 556B. Esta configuración se realiza en las opciones del compilador (CodeVisionAVR v1.24) y queda (Ilustración 21):

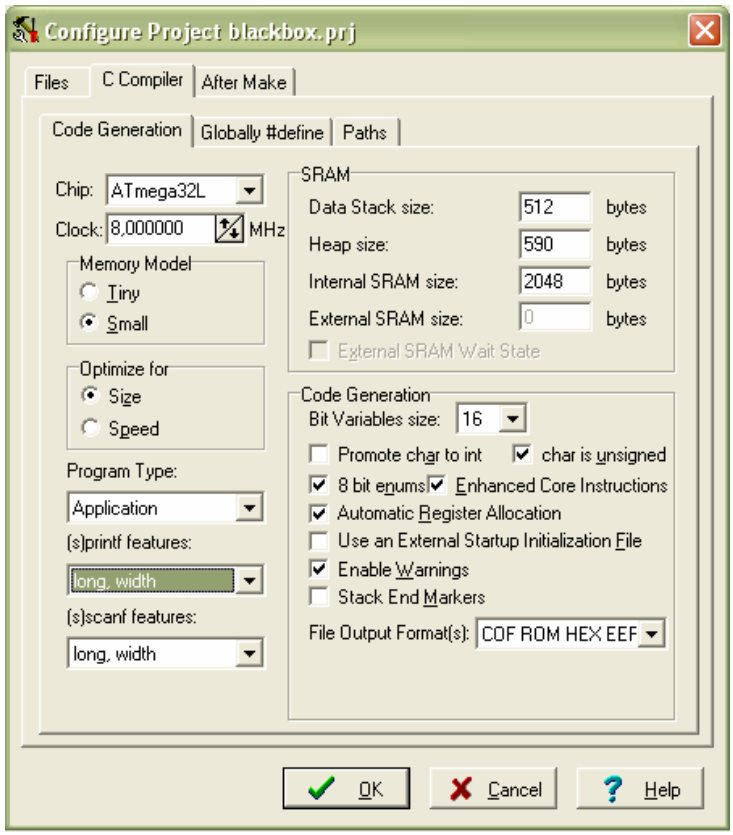


Ilustración 21- Configuración de memoria (Project => Configure => C Compiler => Code Generation)

2.4.3 Opciones de compilación

Todas las opciones de compilación de la librería se encuentran en "options.h" y permiten al usuario reducir el espacio que la librería ocupa en la memoria flash del microprocesador.

ETIQUETA	DESCRIPCIÓN	USO
_options_h_	Evita que el código se incluya varias veces.	En uso
_RTC_ON_	Activa las opciones de tiempo real para impresión de hora y fecha en los archivos.	Desactivado
_SECOND_FAT_ON_	Permite actualización simultánea de las tablas FAT mientras escribe. Si no se usa, se evitan conflictos con el PC.	Desactivado
_FAT12_ON_	Permite el uso de sistema de archivos FAT12.	Activado
_READ_ONLY_	Elimina las funciones de escritura con la consecuente reducción de espacio.	Desactivado

_DEBUG_ON_	Permite una salida por el puerto serie que se puede monitorizar con la HyperTerminal. También permite usar: read_directory() y GetVolID().	Desactivado
CVAVR _ICCAVR_	Solo una de las opciones puede estar activa y permite elegir entre los compiladores: CodeVisionAVR y ImageCraftAVR.	_CVAVR_
SD_CS_ON() SD_CS_OFF() CS_DDR	Definen el estado del pin "selector de chip" del SPI del Atmel AVR.	#define SD_CS_OFF() PORTB |= 0x10 #define SD_CS_ON() PORTB &= 0xEF #define CS_DDR_SET() DDRB |= 0x10
_FF_MAX_FPRINT	Longitud máxima de una cadena en la función fprintf().	40
_FF_PATH_LENGTH	Longitud máxima de la cadena que contiene el "path" del archivo.	15

2.5 Etapa 5 – Comunicación con la Unidad de Medida Inercial.

2.6 En este apartado se explica el código de programa concerniente al almacenamiento de datos. El código completo se incluye en el "Otros recursos no utilizados en este proyecto.

Para información sobre los ADC, los Comparadores analógicos, el TWI, o el JTAG consultar [5].

Anexo II: Código del Sistema de Adquisición de Datos", aunque algunas partes del mismo se incluyen en este apartado para ayudar a la explicación.

2.6.1 El buffer circular.

Un buffer circular es necesario para el correcto trasvase de información entre la IMU y el SAD, porque sin él la diferencia de velocidades entre salida de datos por la vía serie de la IMU y la entrada de los mismos en la tarjeta de memoria SD provocaría la pérdida por el camino de algunos de ellos. El funcionamiento del buffer circular es sencillo y se explica a continuación.

Se declaran una cadena de caracteres (*buffer*) que hará de buffer con la longitud necesaria (*SMAX*) para evitar rebose de la misma y dos punteros, uno lee continuamente las entradas al SAD (*p_in*) y las almacena en memoria RAM del microprocesador y el otro (*p_out*), que le sigue siempre por detrás, va almacenando en un puntero auxiliar (*psc*) todos esos caracteres "escupidos" desde la IMU (*ps++ = *p_out++;) hasta llegar a un número de caracteres mínimo (*MINCHAR*), momento en el que se escribe ese grupo de caracteres finalmente en la tarjeta SD.

```
#define SMAX 320
```

```
#define MINCHAR 30

char buffer[SMAX], *p_in, *p_out;

char *psc;
```

Cada vez que *p_in* o *p_out* hacen una operación de lectura (*p_in++ =UDR;) o escritura avanzan una posición dentro de la cadena *buffer* y cuando llegan al final (buffer + SMAX) se devuelve el puntero al principio (buffer).

```
if(p_out >= buffer + SMAX)

p_out = buffer;
```

En caso de que *p_in* alcanzase a *p_out*, lo que supondría que el primero ha dado una vuelta completa más rápido que el segundo, significaría que la longitud de la cadena (SMAX) es demasiado corta y es necesario modificar el código para aumentarla, además generaría una señal de alarma vía parpadeo de los leds del sistema.

```
void error_rebose(void)

{

        while(1)

        {

        PORTA^=0x0C;

        delay_ms(500);

        }

}
```

La lectura de los datos desde la IMU se hace por medio de la interrupción generada por finalización de recepción. El control del

movimiento de los punteros por la memoria reservada para el buffer se gestiona parte en dicha interrupción y parte en la función encargada de la lectura del carácter entrante (migets).

La Ilustración 22 muestra un diagrama explicativo:

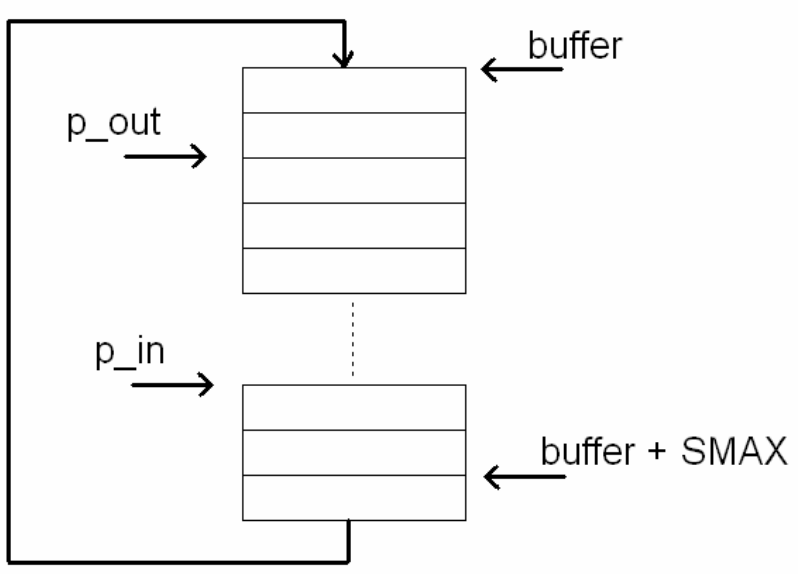


Ilustración 22 – Buffer circular

2.6.2 Programa main.

Lo primero que hace el program principal es inicializar los periféricos del microprocesador. Para ello llama a la función *init_devices()* que configura los puertos de entrada/salida, la velocidad de funcionamiento de la vía serie (38400 baudios) y demás periféricos.

Después pasa a inicializar la tarjeta SD mediante la función *initialize_media()* que es parte de la librería FlashFile y se explicará en el apartado 2.6.3 de este capítulo.

Tras una serie de operaciones de comprobación del estado del sistema, búsqueda del último archivo existente en la tarjeta SD y creación del nuevo archivo (función *fcreate* de FlashFile), se pasa a escribir los datos que entran por la vía serie a la tarjeta SD, por medio del buffer circular ya explicado y la función de FlashFile *fprintf*. Esta etapa termina accionando un interruptor y se pasa a cerrar el archivo mediante *fflush* y *fclose*, que se explicarán en apartados sucesivos.

La Ilustración 23 – Pseudocódigo de la función main muestra el pseudocódigo de lo que se acaba de explicar brevemente con palabras:

```
Inicializar periféricos
Parpadeo en los leds (comprobación de comienzo de programa)
Inicialización de la SD
Led verde ON
Esperar: mientras(jumper escritura ON)
Filtro de rebotes
Led rojo ON
Buscar archivo libre en la SD
Crear archivo siguiente
Activar interrupciones de recepción finalizada
Escritura continua en archivo
Cerrar archivos
Led rojo OFF
```

Ilustración 23 – Pseudocódigo de la función main

Las configuraciones de periféricos del microprocesador se explicarán en el "Anexo I: Guía básica sobre manejo de AVR ATMEL Atmega" y para función de escritura continua en la SD se utiliza el buffer circular que ya se explicó.

Las funciones de las que a continuación se incluye una breve descripción y un sencillo ejemplo se encuentran en los ficheros *"sd_cmd.c"* y *"file_sys.c"* de FlashFile.

2.6.3 Función *initialize_media()* de la biblioteca FlashFile.

La función *initialize_media* no recibe ningún parámetro y devuelve un carácter para comprobación de inicialización correcta. Es necesario llamar a esta función para poder trabajar con la tarjeta de memoria SD o MMC.

unsigned char ***initialize_media(****void****)***

Ejemplo sencillo de utilización:

```
if (initialize_media()) // Si la tarjeta se inicializa correctamente
       putsf("OK");     // imprimir "OK"
else                    // Sino,
       putsf("ERROR");  // imprimir "ERROR"
```

2.6.4 Función *fcreate* de la biblioteca FlashFile.

La función *fcreate* recibe el nombre del archivo a crear y devuelve un puntero a la dirección donde ha sido creado. Todo archivo abierto debe ser cerrado tras su uso.

****FILE* ***fcreate(****unsigned char* ****NAME)***

Ejemplo sencillo de utilización:

```
strcpy(filename, "TEST.TXT"); //nombre del archivo
file1 = fcreate(filename);   // Se crea el archivo "TEST.TXT" en la SD
fclose(file1);               // Cerrar el archivo
```

2.6.5 Función *fprintf* de la biblioteca FlashFile.

La función *fprintf* recibe el puntero que apunta al archivo donde se va a escribir y los datos a escribir y no devuelve nada. El funcionamiento es similar a su equivalente en <stdio.h> de cualquier compilador de C.

void ***fprintf(****FILE* ****rp,*** *unsigned char flash* ****pstr, ...)***

Ejemplo sencillo de utilización:

```
sum = 3 + 4; // sum = 7
fprintf(fp, "The number %d is the answer", sum); // imprimir en
archivo apuntado por fp el valor de sum
```

2.6.6 Función *fflush* de la biblioteca FlashFile.

La función *fflush* recibe el puntero al archivo que debe salvar y devuelve un entero para comprobación de operación de salvado correcta.

int ***fflush(****FILE* ***rp)**

Ejemplo sencillo de utilización:

```
{…}    //operaciones del programa
flag = fflush(fp); // salvar el archivo apuntado fp
```

2.6.7 Función *fclose* de la biblioteca FlashFile.

La función *fclose* recibe el puntero al archivo que debe cerrar y devuelve un entero para comprobación operación de cerrado correcta.

int ***fclose(****FILE* ****rp)***

Ejemplo sencillo de utilización:

```
{…}     //operaciones del programa
flag = fclose(fp); // cerrar el archivo apuntado por fp
```

3 Resultados

Acabado el diseño, montaje, programación y depuración de la unidad se hicieron una serie de pruebas para comprobar el correcto funcionamiento. Una de ellas fue conectar al SAD la IMU y mover el conjunto con la mano. A continuación se muestra parte del contenido del archivo que se grabó en ese ensayo. Las columnas 5, 6 y 7 son aceleraciones lineales en Z, Y y X. Las columnas 2, 3 y 4 son las aceleraciones de rotación en los ejes Z, X e Y.

```
{…}
$GPADC,079,07C,11F,135,184,254,279,06A
$GPADC,07B,07E,11F,136,186,256,27B,06C
$GPADC,07E,080,11E,136,185,25B,275,06F
$GPADC,07E,081,11F,136,185,25D,277,06F
$GPADC,07B,07E,11F,136,184,25E,282,06C
$GPADC,07A,07D,11D,135,188,257,27A,06B
$GPADC,07E,081,11E,137,186,252,27F,06F
$GPADC,07C,07F,11E,136,185,259,288,06D
$GPADC,07D,080,11C,136,187,259,273,06E
$GPADC,07A,07D,11C,138,181,25C,27B,06B
$GPADC,07B,07D,11F,132,18A,264,272,06C
{…}
```

La representación de los valores contenidos en este archivo se hizo con ayuda de MATLAB:

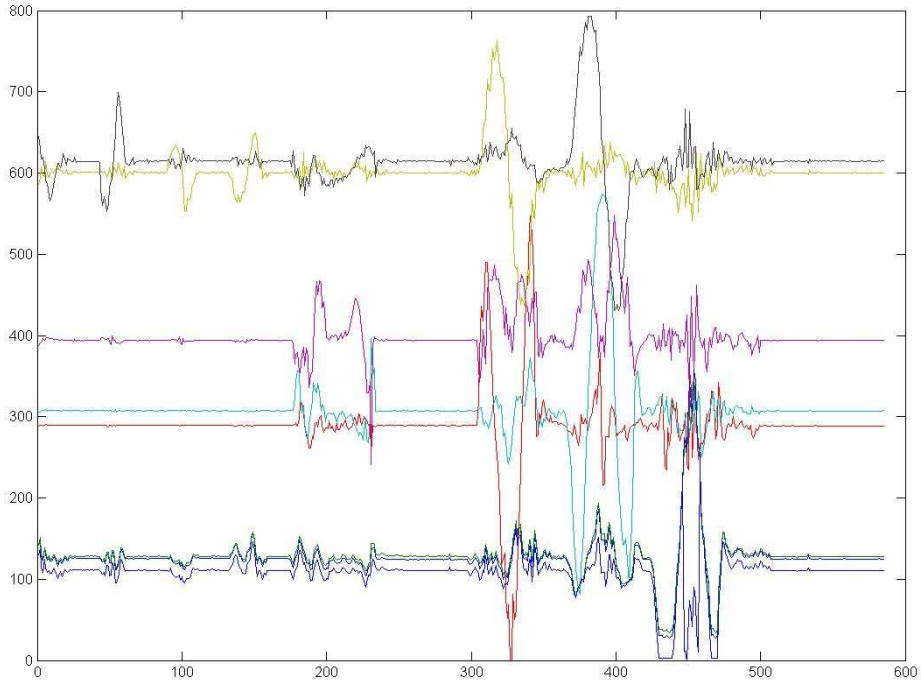


Ilustración 24 – Representación de los ADC. Negro, Verde y Morado: Aceleraciones lineales. Azul, Rojo y Azul Oscuro: Aceleraciones angulares

Donde cada una de las señales que aparecen representadas corresponden con uno de los canales ADC –Analog to Digital Converter- con los que se obtienen las medidas de aceleración lineal y velocidad angular del cuerpo en estudio.

4 Recursos empleados

En esta sección se explican brevemente los recursos empleados para la realización de esta fase del proyecto.

4.1 Eagle

Eagle ya ha sido explicado detalladamente a lo largo de este capítulo. A modo de resumen, solamente decir que es un programa de diseño para elaboración de circuitos electrónicos que permite generar el "layout" de una placa a partir de un esquema del conexionado de componentes que se realiza en un entorno gráfico muy sencillo.

4.2 AVR Dude de Win AVR

Es un programa que permite configurar los "Fuse Bits" de los microprocesadores AVR y cargar un bootloader en los mismos utilizando directamente la vía SPI.

4.3 AVR Studio 4

Es el programa propio de ATMEL Corp. utilizado para compilar el bootloader (en ensamblador) que se utiliza para este microprocesador (AtMega32L).

4.4 Kargador y bootloader

El bootloader utilizado en este proyecto se llama Kargador y esta dividida en dos partes:

- Programa en lenguaje Delphi cedido por Alejando Weinsteinen que se ofreció a compilar una versión del programa para la velocidad de comunicación serie de 38400 baudios, ya que la versión que él tenía disponible no se adaptaba a las necesidades. Esta parte va directamente

instalada en el ordenador. La imagen siguiente muestra la ventana que se abre en la pantalla del ordenador al ejecutar Kargador.

- Programa en ensamblador que se instala en el microprocesador.

En líneas generales el bootloader sigue el siguiente proceso:

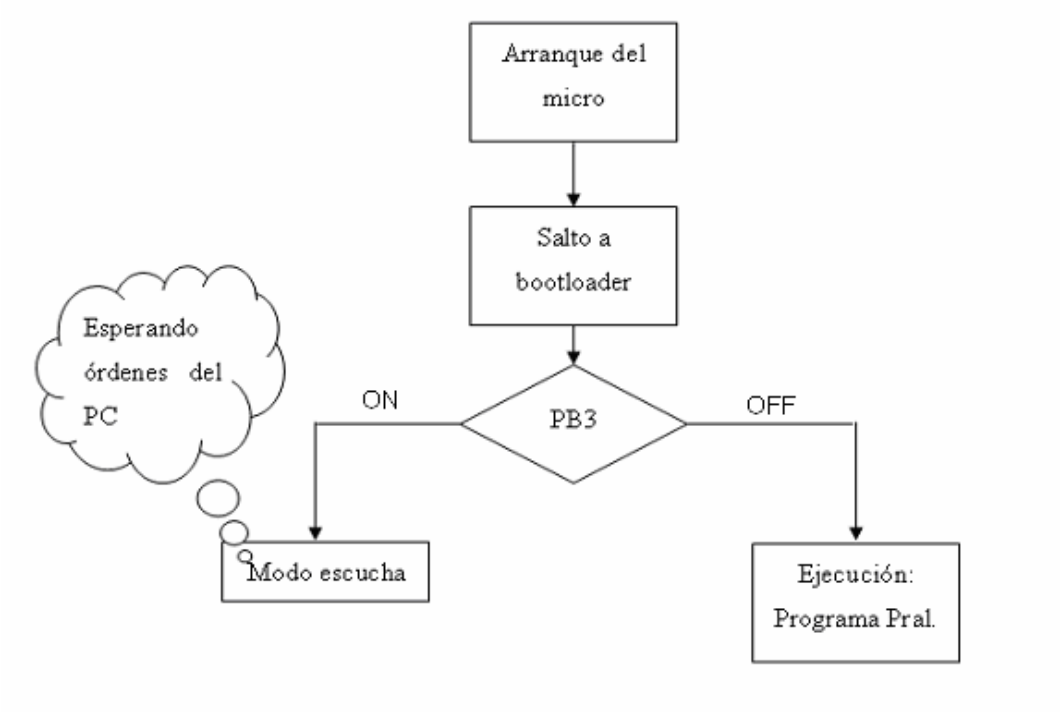


Ilustración 25 – Diagrama de flujo del Bootloader.

Mientras el bootloader está a la espera de órdenes (Modo Escucha). En la ventana del ordenador (Ilustración 26) se eligen los siguientes parámetros:

- Microcontrolador a programar.
- Puerto para la programación.

- Pinchar en “Load File” y seleccionar el archivo compilado a cargar en el micro.
- Pinchar en “Program” y esperar…

Ilustración 26 –Ventana principal de Kargador.

4.5 FlashFile

Es una librería de funciones adquirida en Progresive Resources LLC para el tratamiento de ficheros y archivos en tarjetas SD. Su utilización ha sido explicada ya en los apartados anteriores.

4.6 CodeVision

CodeVision es un compilador de C para desarrollo de ejecutables para microprocesadores de la familia AVR. El entorno de trabajo se muestra en la Ilustración 27.

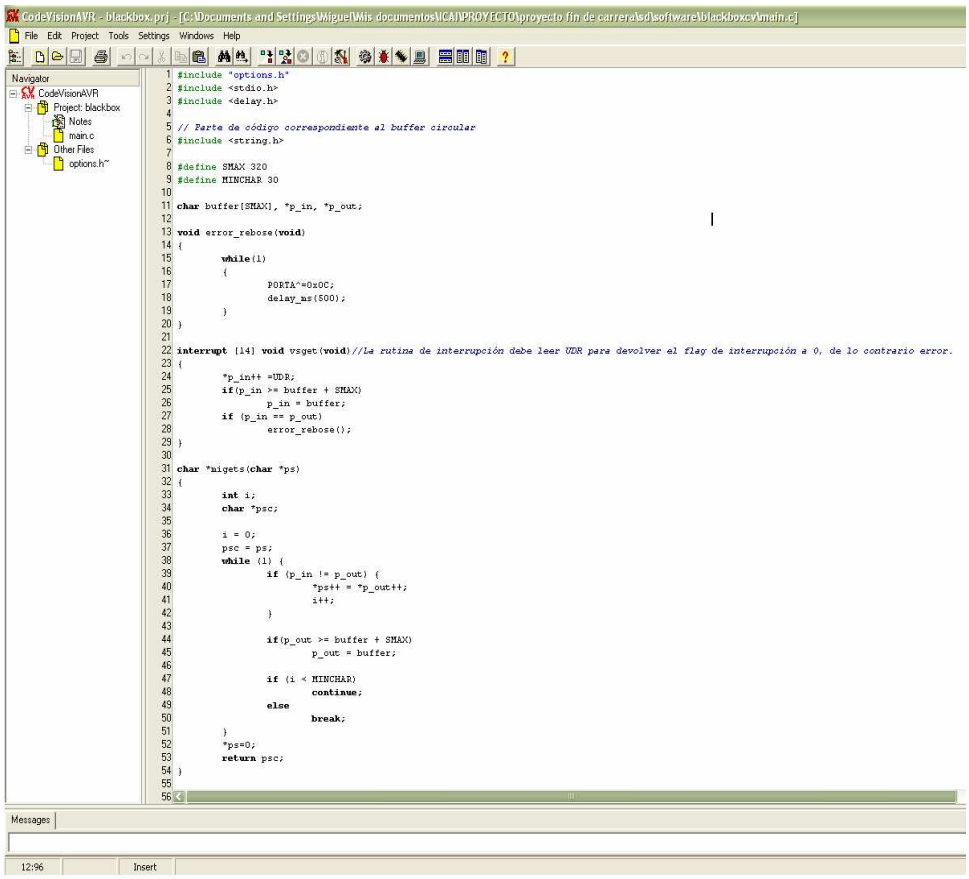


Ilustración 27 – Entorno de trabajo en CodeVision

En la imagen anterior podemos distinguir varias partes:

1. La ventana principal: Es la ventana en la que se tratan los archivos de código (.c y .h).
2. El Navegador: Situado a la izquierda muestra el árbol del proyecto.
3. Barras de herramientas: Botones de acción rápida.

5 Otras posibles aplicaciones

Este dispositivo ha sido diseñado en este proyecto con la única idea de almacenar los datos que expulsa por vía serie la Unidad de Medida Inercial en un soporte de fácil uso (una tarjeta de memoria tipo SD), pero es lógico que cualquier futura aplicación que necesite de una recogida de datos que sean comunicados por una vía serie podría utilizar este módulo con tan sólo unos pocos retoques en el código, si así se creyera necesario.

Capítulo 4 INTEGRACIÓN DE IMU Y SAD.

En este capítulo se va a explicar el proceso de montaje de ambos sistemas dentro de una caja de plástico para su protección.

1 La caja apropiada

El primer paso fue la búsqueda de una caja de dimensiones adecuadas para albergar los módulos y que además fuera lo suficientemente hermética para que no entrasen desde el exterior partículas que pudieses dañar los sistemas que contienen.

La caja que se encontró tiene las siguientes características:

1. Dimensiones (largo-ancho-alto, en milímetros): 150 x 90 x 48
2. Material: Todo de plástico excepto la tapa que es de metal y va atornillada.

2 Ubicación de los módulos

El siguiente paso fue la planificación de espacio para determinar las posiciones de las diferentes placas, teniendo en cuenta que el eje Z de la IMU tiene que estar totalmente vertical, que los ejes X e Y de la IMU tienen que estar horizontales y que la ranura del zócalo SD tiene que estar pegada a una de las paredes de la caja para su fácil extracción (ver Ilustración 28)

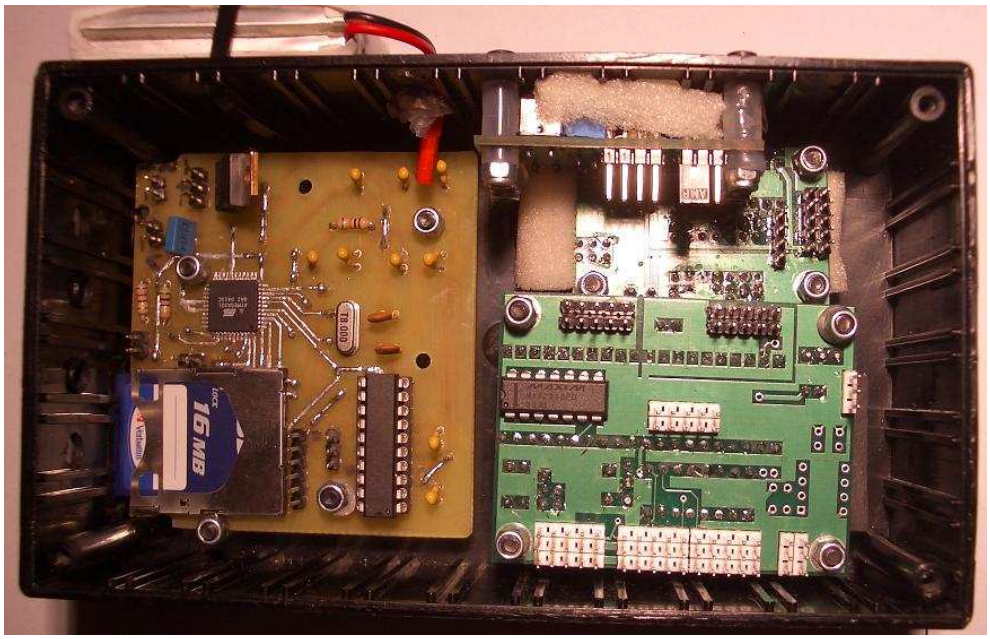


Ilustración 28 – Colocación de los 4 módulos

3 Sistema de aislamiento

Una vez determinada la posición de cada uno de los componentes se colocó un aislamiento de goma-espuma en los módulos de medida XY y Z de la IMU y aislamiento de neopreno en la placa de control de la IMU con el fin de aislar en la medida de lo posible las vibraciones que pudieran aparecer durante la utilización del módulo (ver Ilustración 29).

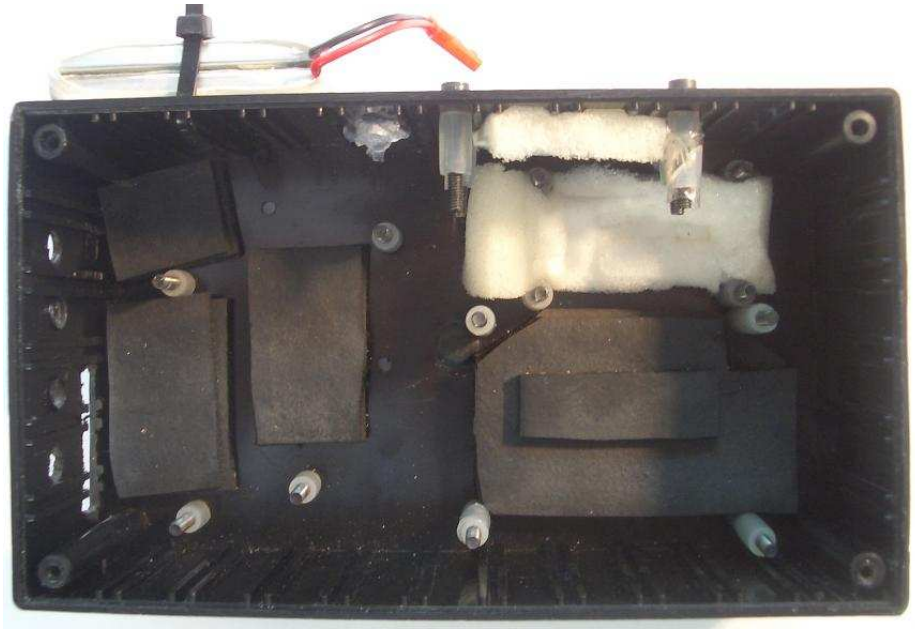


Ilustración 29 – Aislamiento

4 Sistema de sujeción

Para fijar cada uno de los módulos convenientemente se han utilizado tornillos con "relleno" de silicona como muestra la Ilustración 30:

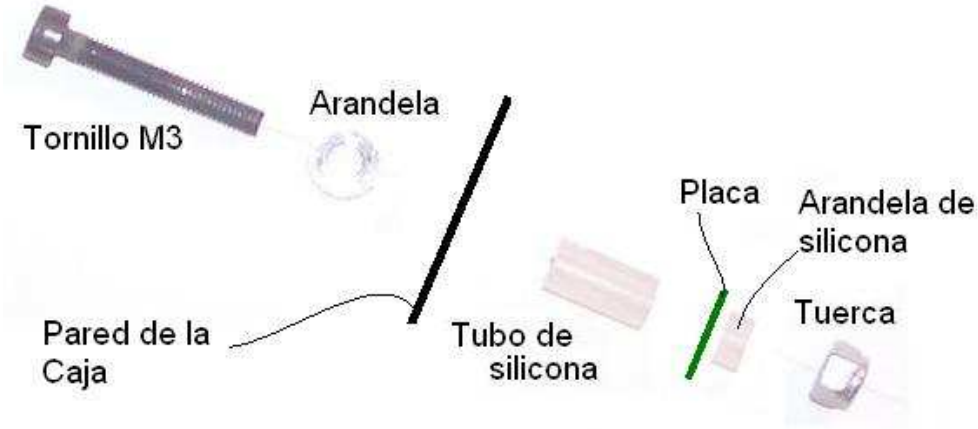


Ilustración 30 – Sistema de Sujeción

Todos los tornillos son de métrica M3, cabeza tipo Allen y de 2 centímetros de longitud, excepto los que sirven para la sujeción de la placa de control de la IMU que son de 3 centímetros puesto que esta placa está colocada "flotando" sobre la placa de medida XY de la IMU.

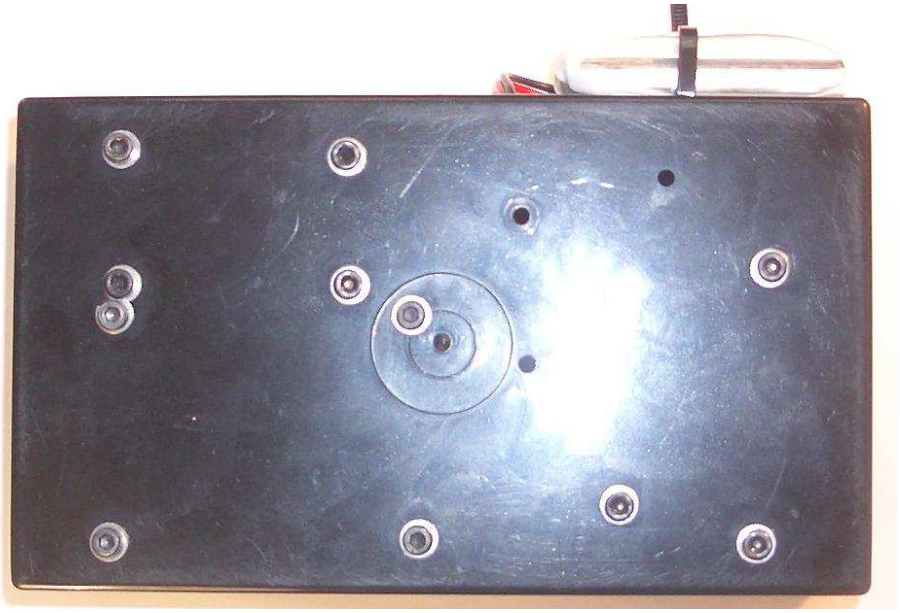


Ilustración 31 – Taladros

Las tuercas son auto-blocantes para mayor seguridad.

5 Panel de mandos

La Ilustración 32 muestra el panel de mandos, compuesto por: interruptor de encendido/apagado, interrupción de grabación, led de ESTADO, led de ON/OFF y ranura de extracción.

Ilustración 32 – Panel de Mandos de IMU+SAD

6 Proceso de montaje paso-a-paso

El proceso de montaje del sistema integral IMU+SAD pasa por las siguientes fases:

Paso 1: Colocación de las 4 placas.

- Placa de Medida en los ejes XY: Se coloca con la cara inferior hacia arriba y con los conectores negros hacia la pared posterior.
- Placa de Control de la IMU: También con la cara inferior hacia arriba y con el banco de conectores blancos pegados a la pared izquierda.
- Placa de Medida en los ejes Z: La cara superior debe estar pegada contra la pared derecha y con los conectores negros hacia arriba.
- Placa SAD: La cara de componentes hacia arriba y con el Zócalo para la tarjeta SD pegado a la ranura de la pared delantera (panel de mandos).

Una imagen de la colocación de los 4 módulos puede verse en el apartado 2 de este capítulo.

Paso 2: Fijar las placas con los tornillos.

El sistema de fijación ya ha sido comentado en el apartado 4 de este capítulo. Las herramientas necesarias son una llave allen para tornillos de métrica 3 y un alicate de punta fina para sujetar las tuercas.

Paso 3: Colocación de los LEDS.

El led verde se coloca en el conector de dos pines marcado con una "V" y el rojo en el marcado como "R". El polo positivo (+) está marcado tanto en la placa como en el conector de los leds.

Una vez enchufados se colocan en los taladros practicados para ello en el panel de mando. El verde se saca por el marcado como "POWER" y el rojo por "STATUS". Es conveniente fijarlos con silicona.

Paso 4: Colocación de los interruptores (ver Ilustración 32)

Para el interruptor de "POWER" (que es el grande de los dos) se ha construido un cable que da alimentación a la Placa de Control y al SAD, y recibe un cable desde la batería.

El interruptor de "STATUS" (el pequeño) se coloca en el conector de dos pines cercano a la ranura de la SD con la flecha en el pin de la izquierda y se fija al panel de mandos a través del taladro practicado para ello.

Paso 5: Conexión de los demás cables.

Las conexiones de los diferentes módulos de la Unidad de Medida Inercial ya se vieron en el apartado 2.2 del Capítulo 2 (ver Ilustración 6– Placa de Control. Cara inferior, Ilustración 7 – Placa XY e Ilustración 8 – Placa Z).

El cable serie que va desde la Placa de Control de la IMU al SAD se conecta en la IMU en el banco de conectores blancos en el marcado como

RS-232 con el cable pintado de negro en el punto negro respectivo y en el SAD según la Ilustración 33.

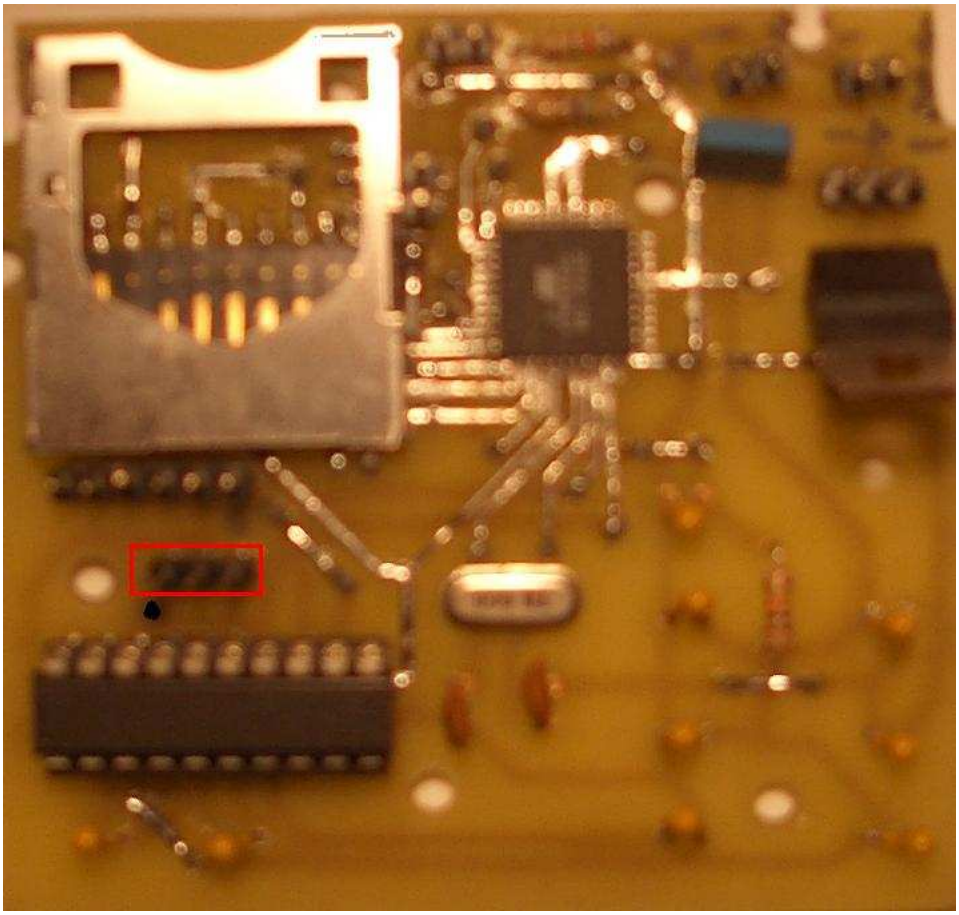


Ilustración 33 – Cable Serie del SAD

Los cables de alimentación salen del interruptor de alimentación y deben conectarse en:

- La batería.
- La Placa de Control de la IMU (Ilustración 34).
- El SAD (Ilustración 35).

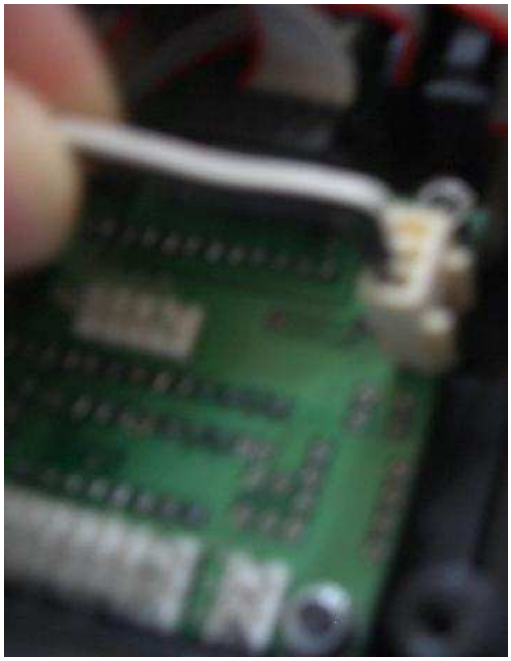


Ilustración 34 – Alimentación IMU

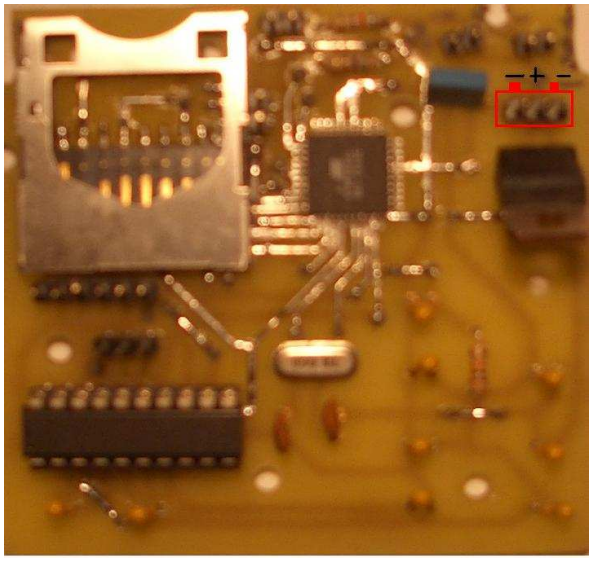


Ilustración 35 – Alimentación SAD

Paso 6: Poner la tapa y atornillar

7 Ejes

Los ejes del conjunto así obtenido quedan como se indican en la Ilustración 36 – Los ejes del SAAD.

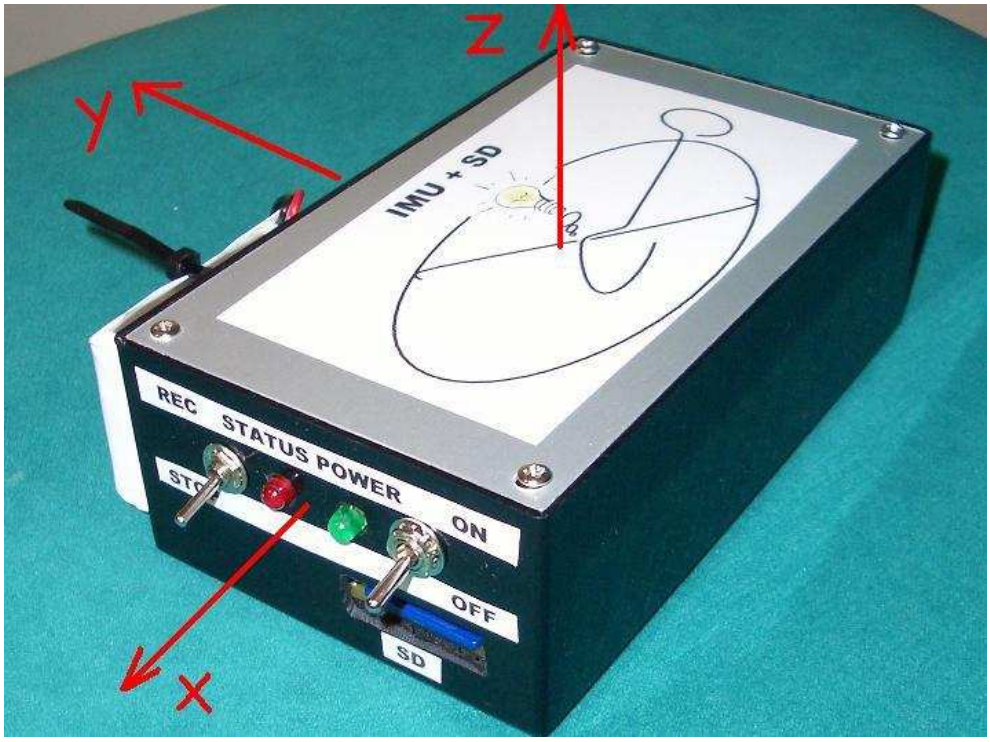


Ilustración 36 – Los ejes del SAAD

Capítulo 5 RECEPTOR

La IMU tiene la posibilidad de lanzar por vía serie, además de las medidas de los sensores, una cadena con los PPM recibidos de la emisora. Esto es muy útil porque en futuros análisis de datos para la realización de un Modelo Dinámico del aparato será necesario relacionar Entradas y Salidas del Sistema. Las salidas son las medidas de los sensores y las entradas, los PPM. Así, en este capítulo se explica como se ha "trucado" un receptor comercial para robar su señal PPM y poder almacenarla en el SAD.

El receptor trucado es un Micro-Receptor ACT de 6 canales que se muestra en la Ilustración 37 - Receptor.

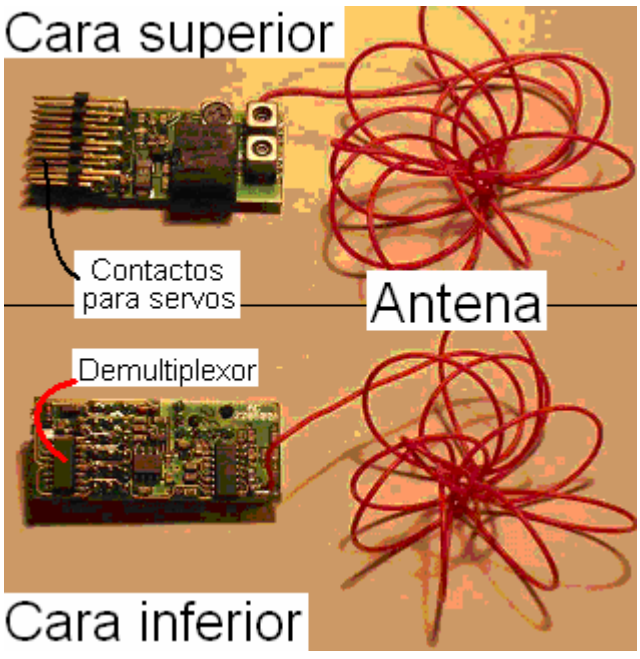


Ilustración 37 - Receptor

Lo necesario para robar la señal del tren de pulsos PPM es encontrar un demultiplexor porque es ahí donde se realiza la separación de los canales de la señal de la emisora antes de mandarlos demultiplexados a los distintos servos conectados al receptor. En la Ilustración 38 – Señales "robadas" del Receptor se puede ver el pin del demultiplexor donde entra la señal PPM de la que se ha estado hablando; además se han marcado dos puntos (V- y V+) para colocar la alimentación del dispositivo.

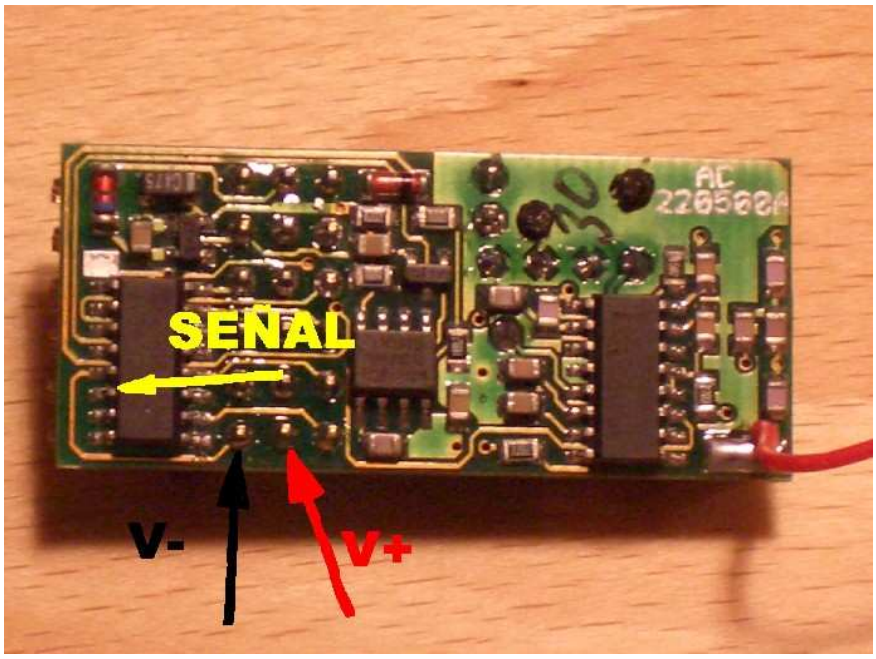


Ilustración 38 – Señales "robadas" del Receptor

La SEÑAL "robada" se pasa conecta a la Placa de Control de la IMU en el conector marcado como 'P'.

El siguiente fragmento es parte de una prueba que se hizo con el receptor conectado al sistema:

{…}
$GPADC,7FE0,7FE0,7FE0,5E5F,6CDE,5959,5112,47D6
$GPPPM,2F1C,2F71,3BD7,3926,40CA,4052
$GPADC,7FE0,7FE0,7FE0,5E60,6CE7,595A,510E,47D8

$GPPPM,2F1D,2F70,3BD8,3925,40CB,4051
$GPADC,7FE0,7FE0,7FE0,5E57,6CFA,595C,511A,47D5
$GPPPM,2F1C,2F71,3BD7,3925,40CB,4053
$GPADC,7FE0,7FE0,7FE0,5E5F,6CF0,595E,5116,47D9
$GPPPM,2F1C,2F71,3BD7,3925,40CB,4053
$GPADC,7FE0,7FE0,7FE0,5E54,6CF0,595B,5114,47D7
$GPPPM,2F1C,2F71,3BD8,3924,40CB,4052
$GPADC,7FE0,7FE0,7FE0,5E50,6CED,595E,511A,47D5
$GPPPM,2F1C,2F71,3BD8,3925,40CB,4052
$GPADC,7FE0,7FE0,7FE0,5E60,6CDF,595D,511F,47D8
$GPPPM,2F1D,2F70,3BD7,3925,40CB,4052

Capítulo 6 CONCLUSIONES.

La Unidad de Medida inercial analizada y modificada en el Capítulo 2 permite la adquisición de las medidas necesarias para la determinación de la posición de un cuerpo en tres dimensiones. A parte, otras muchas utilidades están incluidas con este módulo adquirido a la empresa Rotomotion LLC. Si bien es cierto que dichas utilidades no permiten más que un control muy sencillo del helicóptero para estabilizarlo en vuelo estacionario.

Comparado con otros sistemas del mercado que permiten hacer medidas similares, es mucho más barato, pero también es necesario decir que es, seguramente, mucho menos preciso y versátil, a parte de incluir menor número de sensores integrados; otros sistemas poseen GPS, brújula 3D,…

La mayor ventaja que se puede extraer del uso de esta IMU y no de otra es la posibilidad de estudiarla a fondo gracias a la licencia GNU que tiene el producto. Este hecho nos proporciona la posibilidad de aprender más y más rápido que si hubiéramos hecho el desarrollo desde cero, además de aprender de los errores de los demás.

Aunque el estudio de las utilidades no relacionadas directamente con la adquisición de datos no ha sido tan profundo como el de las que si lo estaban, ha sido suficiente como para dejar las bases, sobre todo

documentales, que permitan en el futuro investigar en otras líneas, como los Filtros de Kalman y la identificación de parámetros en el Modelo Dinámico.

En cuanto al Sistema de Adquisición de Datos se debe decir que es muy versátil y con grandes posibilidades, porque su utilización en cualquier tipo de almacenamiento a través de vía serie es fácilmente realizable gracias a unos pequeños retoques en el código (ver Parte III Manual de usuario).

La combinación de estos dos módulos, IMU y SAD, proporciona un sistema de sencillo uso como se podrá concluir tras la lectura de la Parte III Manual de usuario. La intención del proyecto ha sido su uso en helicópteros a escala, pero su aplicación a todo cuerpo en movimiento es evidente.

Tras la realimentación generada por los ensayos realizados en la segunda parte del proyecto, se ha llegado a la conclusión de que seria conveniente rediseñar la disposición de los diferentes módulos para que la influencia en las medidas fuera menor. La línea de investigación debería ir por una dispersión adecuada de los sensores en el chasis del helicóptero, en vez de la caja creada en el Capítulo 4.

BIBLIOGRAFÍA

Bibliografía principal

[1] Rotomotion, "Guía de Montaje de la IMU" www.rotomotion.com/rev2.4.build/rev2.4_build_toc.html

[2] Analog Devices, "ADXL202 Datasheet", www.analog.com/

[3] Autopilot, autopilot.sourceforge.net

[4] Progresive Resources LLC, "Guía de uso de Flash File", www.prllc.com/

[5] ATMEL ,"Manual Atmega-32/32L"

Bibliografía complementaria

[6] Buskey, G., Roberts, J., Corke, P. and Wyeth, G., "Helicopter Automation Using a Low-Cost Sensing System", Australasian Conference on Robotics and Automation, 2003

[7] Buskey, G., Roberts, J., Corke, P. and Wyeth, G., "Sensing and Control for a Small-Size Helicopter", International Symposium on Experimental Robotics, Italy, 2002

ANEXO I: GUÍA BÁSICA SOBRE MANEJO DE AVR ATMEL ATMEGA

Esta guía pretende explicar el manejo básico de los microprocesadores utilizados en el desarrollo del proyecto. En los ejemplos y para los valores concretos se utilizará como referencia el ATMEL AtMega32-L, por lo tanto, es posible que, en algunos puntos, si se utiliza otro microprocesador de la familia (AtMega-163, AtMega-8515, etc.), los ejemplos no sean válidos.

1 Características generales

- La familia AtMega está compuesta por microprocesadores con registros y ALU de 8 bits.
- Pensados como microprocesadores de propósito general.
- Disponen de memorias RAM, EEPROM y Flash internas.
- Integran en el chip un amplio abanico de periféricos:
  - Controladores de comunicación serie (SPI y USART)
  - Varios puertos paralelo.
  - Varios Timers/Counters (8 y 16 bits)
  - Comparador analógico.
  - Conversor Analógico/Digital (ADC)

- Moduladores PWM para electrónica de potencia.

2 Programación

Los AtMegas se pueden programar tanto en lenguaje ensamblador como en lenguaje C. En este documento se utilizará el segundo de ellos. Un completo juego de instrucciones puede encontrarse en

3 Arquitectura del microprocesador

La Ilustración 39 muestra un esquema general de la arquitectura interna del microprocesador.

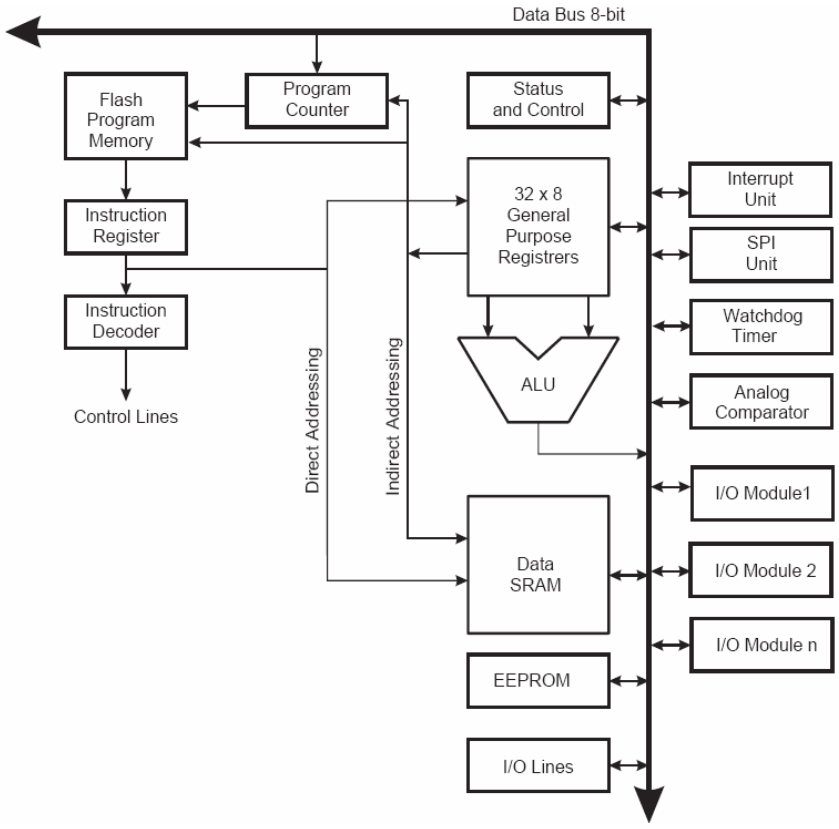


Ilustración 39 – Arquitectura del microprocesador

4 Utilidades

4.1 Puertos I/O. Funciones alternativas.

El AtMega 32L tiene 4 puertos paralelo de 8 bits para entrada-salida, desde PORTA hasta PORTD. Además de la función típica de entrada-salida, estos puertos pueden ser configurados para realizar otras funciones alternativas. Así, por ejemplo, el puerto A puede utilizarse como Conversor Analógico-Digital (ADC), el puerto B tiene funciones de Interfaz Serie (SPI), Comparador Analógico y entradas de interrupción externa, el puerto C lleva un par de osciladores, las funciones JTAG para depuración de programas y el TWI –Two Wire Interface- y por último el puerto D lleva las salidas de los Timers, un par de generadores de interrupción externa y el TX/RX del USART.

Las instrucciones básicas de manejo de estos puertos para entrada/salida son:

- Para configurar la dirección de los pines del puerto: "DDRx". Puesto a '1' significa SALIDA y a '0' significa ENTRADA.
- En un pin de SALIDA se ESCRIBE mediante el comando "PORTx"
- En un pin de ENTRADA se LEE mediante el comando "PINx".

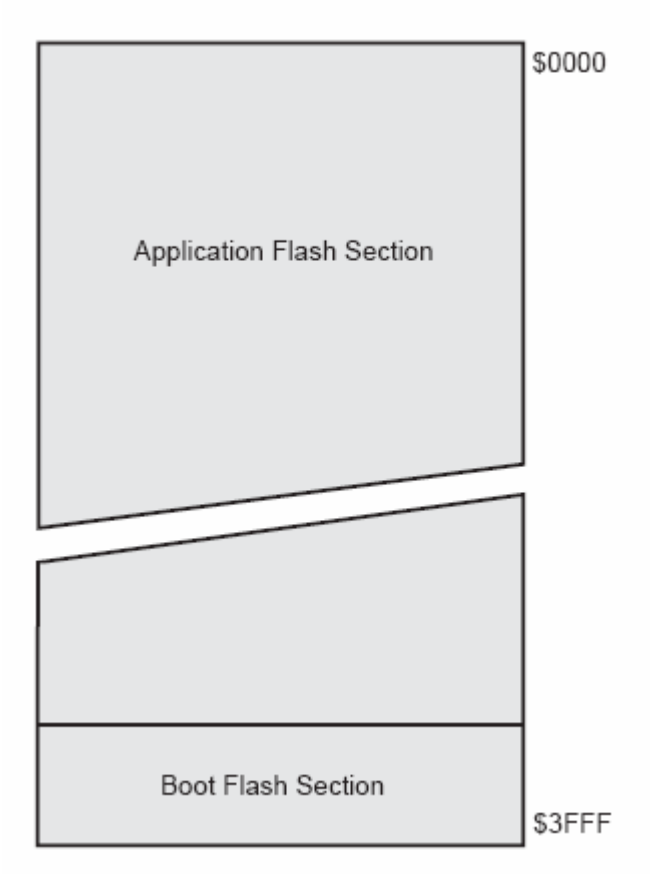
Donde "x" es el puerto (A, B, C o D).

Si uno se quiere referir a una posición concreta del puerto, por ejemplo, para poner un '1' en dicha posición, es una forma sencilla utilizar

algo como esto: “PORTx = (1<<Pxn)”; donde ‘n’ es la posición del puerto donde quieres escribir el ‘1’.

4.2 Memoria.

El mapa de memoria de este micro es el siguiente:



La Sección de Bootloader es variable y se configura con los Fuse Bits como se hizo en el apartado 2.3 del capítulo 3 según BOOTSZ1-0 en la tabla siguiente:

Table 105. Fuse High Byte

Fuse High Byte	Bit No.	Description	Default Value
OCDEN[4]	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN[5]	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN[1]	5	Enable SPI Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT[2]	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 100 for details)	0 (programmed)[3]
BOOTSZ0	1	Select Boot Size (see Table 100 for details)	0 (programmed)[3]
BOOTRST	0	Select reset vector	1 (unprogrammed)

También es conveniente echar un vistazo a los "Bootloader Lock Bits" que permiten la protección de algunas secciones de la memoria según las necesidades del usuario.

4.3 Vía SPI –Serial Periferial Interface-.

Es una vía de comunicación serie síncrona que permite una conexión rápida con el exterior u otros AVR.

Existen dos posibles modos de funcionamiento: Modo Maestro –Master, en inglés- y Modo Esclavo –Slave, en inglés-, que definen si el AtMega es o no es quien maneja las comunicaciones en la SPI.

Una comunicación pasa por las siguientes etapas:

- El maestro de la SPI inicia la comunicación poniendo a cero, por software, el pin SS/ -Slave Select- del esclavo requerido.
- En maestro o esclavo, según modo de configuración, se colocan los datos en los Registros de Intercambio –SPDR-
- El maestro genera la señal de reloj necesaria (sincronismo).

- La comunicación fluye por las líneas MOSI o MISO según esta vaya de maestro a esclavo o viceversa. Al terminar se pone a uno la Bandera de Fin de Transmisión –Transmission Flag (SPIF)- que generará una interrupción si SPIE está activo.
- Para continuar mandando datos el maestro coloca otro en SPDR, mientras que si se quiere dar por concluida la comunicación el maestro re-sincroniza con el esclavo poniendo a uno el pin SS/.

Nota importante: Todo dato a transmitir no puede ser escrito al SPDR antes de que un ciclo de intercambio haya sido concluido, mientras que todo dato recibido debe ser leído antes de que el siguiente entre.

MOSI, MISO, SCK, y SS/ se configuran según la siguiente tabla:

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Los registros necesarios para la configuración y funcionamiento del SPI son:

Registro de Control:

7	6	5	4	3	2	1	0	
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR

SPIE = Para habilitar interrupciones a ‘1’

SPE = Para habilitar dispositivo a ‘1’

DORD = ‘1’ si LSB se transmite primero y a ‘0’ si MSB se transmite primero.

MSTR = ‘1’ si Maestro.

SPR1-0 = Configura la frecuencia del dispositivo según:

Table 58. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Registro de Estado:

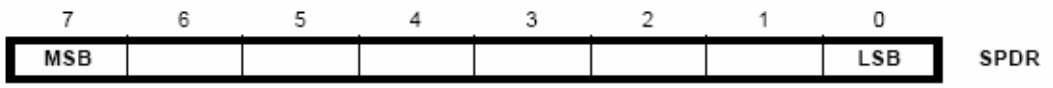
7	6	5	4	3	2	1	0	
SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR

SPIF = Flag de interrupción. Se pone a ‘1’ si se ha completado una transferencia.

WCOL = Se pone a ‘1’ si SPDR se escribe durante una transferencia.

SPI2X = Si está a ‘1’ se dobla la frecuencia en Modo Maestro.

Registro de Datos: Inicia la escritura cuando lo escribes y la actualización del buffer si lo lees.



A continuación se muestran ejemplos simples de inicialización, transmisión y recepción de datos en código C:

```
void SPI_MasterInit(void)
{
/* MOSI y  SCK como salidas. El resto como entradas */
DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
/* Activar interrupción, modo Maestro y frecuencia de reloj*/
SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}
void SPI_MasterTransmit(char cData)
{
/* Comienzo de transmisión */
SPDR = cData;
/* Esperar por POOLING */
while(!(SPSR & (1<<SPIF)));
}
void SPI_SlaveInit(void)
{
/* MISO como salida. El resto como entradas */
DDR_SPI = (1<<DD_MISO);
/* Activar via SPI */
```

```
SPCR = (1<<SPE);

}

char SPI_SlaveReceive(void)

{

/* Esperar por POOLING */

while(!(SPSR & (1<<SPIF)));

/* Devolver dato*/

return SPDR;

}
```

4.4 USART –Universal Syncronous Asyncronous Reciever Transmiter-

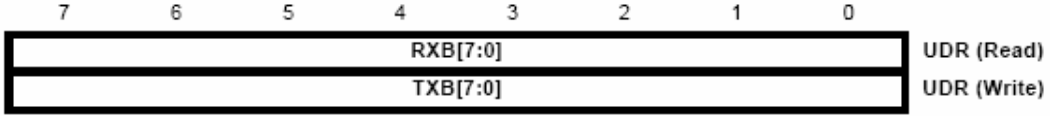
Es un recurso muy flexible de comunicación vía serie ya que permite comunicación síncrona o asíncrona, múltiples "baud rates", ancho de carácter configurable y detección de errores de transmisión.

Cada carácter de transmisión consta de:

- 1 bit de comienzo.
- 5, 6, 7, 8 ó 9 bits de datos.
- 1 bit de Paridad.
- 2 bits de Stop (nivel alto).

Los registros de configuración y funcionamiento del USART son los siguientes:

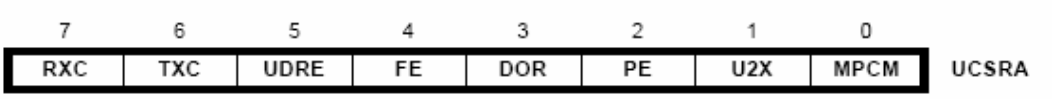
Registro de Datos:



RXB = Buffer de lectura.

TXB = Buffer de escritura.

Registros de Control:



RXC = A ‘1’ si hay datos sin leer del RXB. Puede usarse para generar interrupciones.

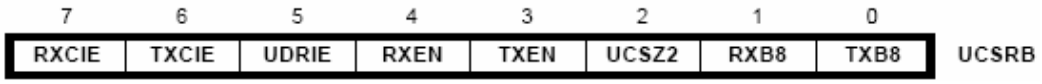
TXC = A ‘1’ si la transmisión se ha completado. Puede usarse para generar interrupciones.

UDRE = A ‘1’ si UDR está vacío. Puede usarse para generar interrupciones.

FE, DOR y PE = Control de errores de “frame”, sobrecarga y paridad, respectivamente

U2X = A ‘1’ dobla la velocidad de transmisión en modo asíncrono.

MPCM = Habilita el Modo de Comunicación de Multi-Procesador.

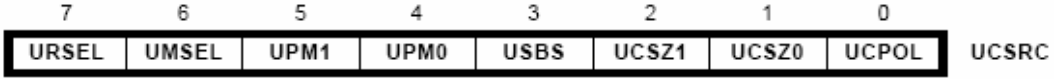


RXCIE, TXCIE y UDRIE = Habilita interrupciones en RXC, TXC y UDRE.

RXEN y TXEN = Habilita el receptor y transmisión.

UCSZ2 = Tamaño del carácter.

RXB8 y TXB8 = El noveno carácter en recepción o transmisión.



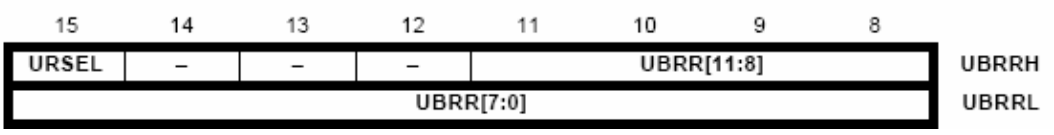
UMSEL = Selecciona entre Modo Síncrono (1) o Asíncrono (0).

UPM1-0 = Modo de paridad según:

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

UCSZ1-0 = Tamaño de carácter.

Registro de Selección de Baud Rate:



URSEL = Selecciona entre acceder a UBRRH (‘0’) o UCSRC (‘1’).

UBRR11-0 = Selección de Baud Rate según el Cristal que genera el reloj, por ejemplo, para un cristal de 8 MHz:

Baud Rate (bps)	f_{osc} = 8.0000 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%
4800	103	0.2%	207	0.2%
9600	51	0.2%	103	0.2%
14.4k	34	-0.8%	68	0.6%
19.2k	25	0.2%	51	0.2%
28.8k	16	2.1%	34	-0.8%
38.4k	12	0.2%	25	0.2%
57.6k	8	-3.5%	16	2.1%
76.8k	6	-7.0%	12	0.2%
115.2k	3	8.5%	8	-3.5%
230.4k	1	8.5%	3	8.5%
250k	1	0.0%	3	0.0%
0.5M	0	0.0%	1	0.0%
1M	–	–	0	0.0%
Max (1)	0.5 Mbps		1 Mbps	

Otras tablas como esta se pueden encontrar en [5].

Algunos ejemplos sencillos de manejo del USART:

```
void USART_Init( unsigned int baud )
{
/* Set baud rate */
UBRRH = (unsigned char)(baud>>8);
UBRRL = (unsigned char)baud;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 2stop bit */
UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}
```

```
void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) )
;
/* Put data into buffer, sends the data */
UDR = data;
}

unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !(UCSRA & (1<<RXC)) )
;
/* Get and return received data from buffer */
return UDR;
}
```

4.5 Timers.

Este micro tiene tres timers:

- Timer/Counter0 de 8 bits.
- Timer/Counter1 de 16 bits.
- Timer/Counter2 de 8 bits.

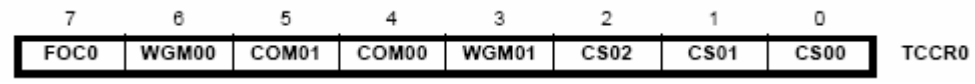
Se explicará el primero de ellos por ser el que se ha usado en este proyecto. Los demás son análogos.

Además de los modos de funcionamiento con PWM explicados en el apartado siguiente, existen otros dos modos:

- Modo Normal: TCNT0 realiza la cuenta desde 0x00 (BOTTOM) hasta 0xFF (MAX), es decir, la cuenta completa.
- Modo CTC: TCNT0 realiza la cuenta desde 0x00 (BOTTOM) hasta TOP (valor que se define en el registro OCR0).

A continuación se pasan a describir los principales registros del Timer0.

Registro de Control:



FOC0 = Se fuerza un Fin de Conteo en uno de los modos no-PWM.

WGM00-01 = Selección de modo, según:

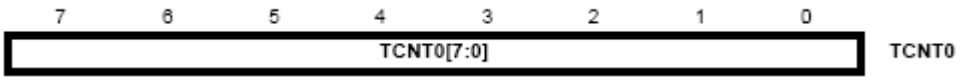
Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

COM00-01 = Configura el pin de salida OC0 de forma que se ponga a ‘1’ o a ‘0’ o cambie de valor cada vez que se de un Final de Conteo (ver [5]).

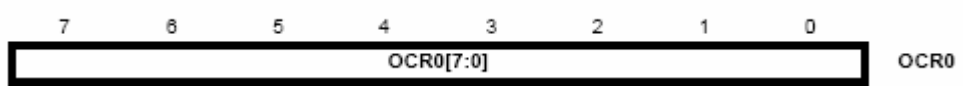
CS00-02 = Configura el pre-escalado del Timer0. Advertir que el pre-escalado es el mismo para los tres Timers, por lo tanto, si lo fijas para uno, queda fijado para los demás. Las posibilidades de pre-escalado son las siguientes:

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

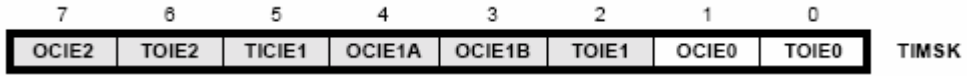
Registro de Conteo: El que lleva la cuenta.



Registro de Valor TOP: Con el que se compara.

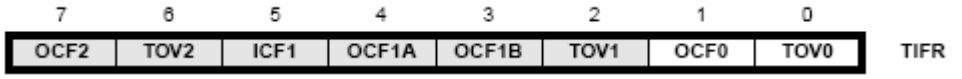


Registro de Habilitación de Interrupciones:



OCIE0 y TOIE0 = A ‘1’ habilitan la interrupción por Comparación y Rebose, respectivamente

Registro de Banderas:

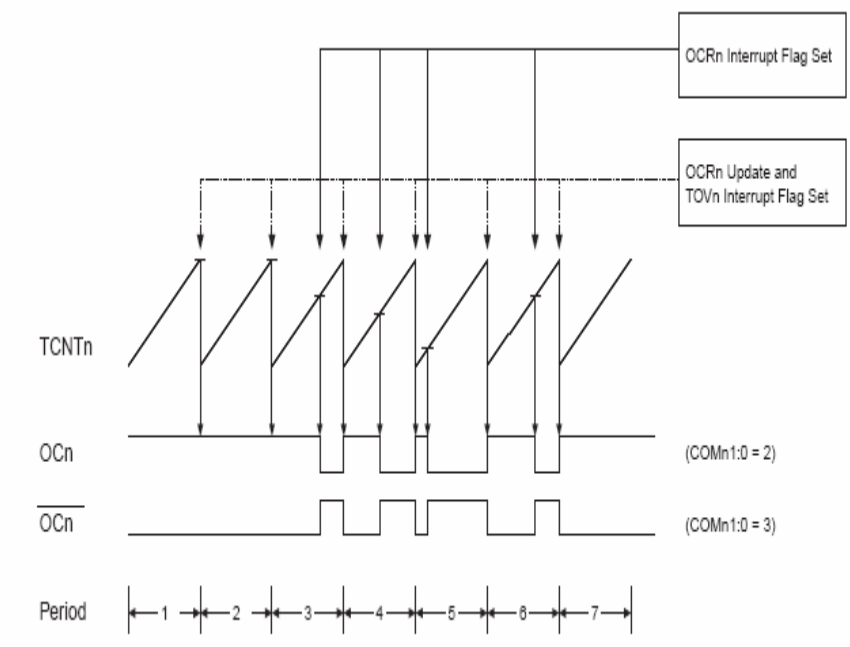


OCF0 y TOC0 = Se ponen a ‘1’ si se produce una Comparación o un Rebose, respectivamente. Se pueden usar como generadores de interrupción.

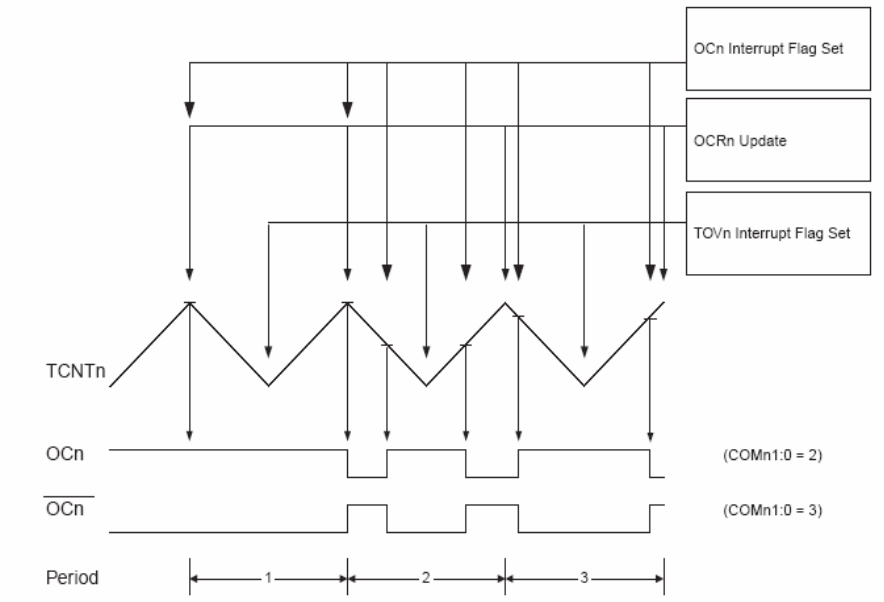
4.6 PWM.

Los PWM son una utilidad del Timer0. Tienen dos formas de operación: “Fast PWM” y “Phase Correct PWM”.

La primera (Fast PWM) es muy apropiada para regulación de potencia y rectificación dada su gran frecuencia. En el pin OCn se genera una señal PWM según el momento en el que los valores OCR0 (registro de comparación) y TCNT0 (registro de conteo) se igualen. Después de este momento el timer acaba la cuenta (MAX), OCn vuelve a su valor inicial y el contador (TCNTn) se reinicia (ver imagen). Para más información sobre los registros ver el apartado 4.5 de este anexo.



La segunda (Phase Correct PWM) provee una señal muy precisa, aunque de menor frecuencia, por ello es útil para control de motores. El modo de funcionamiento difiere del anterior en que primero se realiza la cuenta hacia delante (MAX) y luego hacia atrás (BOTTOM) y el cambió de valor en OCn se produce entre TCNTn = OCRn en bajada y TCNTn = OCRn en subida (ver imagen).



4.7 Interrupciones.

Las interrupciones pueden generarse por múltiples razones como puede observarse en la tabla siguiente que muestra el "vector de interrupciones":

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

La gestión de las mismas necesita que:

- Estén correctamente configurados los bits de Habilitación (en inglés, Enable) de la interrupción concreta.

- Se escriba la "función de interrupción" a la que se acudirá cuando esta se produzca:

```
interrupt [X] void nombre(void)

{

//Aquí se escribe lo queremos que haga el programa cuando salte la interrupción

}
```

Donde 'X' es el número de la interrupción concreta en el vector de interrupciones y 'nombre' es el nombre que se le quiere dar a esta interrupción.

4.8 Otros recursos no utilizados en este proyecto.

Para información sobre los ADC, los Comparadores analógicos, el TWI, o el JTAG consultar [5].

ANEXO II: CÓDIGO DEL SISTEMA DE ADQUISICIÓN DE DATOS

En este anexo se incluye el código realizado para el control del SAD (main.c).

```
#include "options.h"  //Librería de configuración de opciones de FlashFileSD

#include <stdio.h>    //Librería estándar de control de entrada/salida

#include <delay.h>    //Librería de generación de retrasos/retardos


//***************************************************************************
*********

// Parte de código correspondiente al buffer circular

#include <string.h>   //Librería para la gestión de cadenas de caracteres


#define SMAX 320      //Máxima longitud del buffer

#define MINCHAR 30    //Tamaño del grupo de caracteres que se graban cada
vez en la SD


//Declaración de varibles

char buffer[SMAX], *p_in, *p_out;
```

```
void error_rebose(void)      //Función a la que se llama cuando se produce un rebose del buffer circular…
{                    //…A modo de alarma genera un parpadeo cada 1/2 segundo en los LEDS
       while(1)
       {
              PORTA^=0x0C;
              delay_ms(500);
       }
}

interrupt [14] void vsget(void)      //La rutina de interrupción debe leer UDR…
{                           //…para devolver el flag de interrupción a 0; de lo contrario, error.

*p_in++ =UDR;         //UDR (USART Data Register)
if(p_in >= buffer + SMAX)     //Al llegar al final del buffer…
p_in = buffer;        //…se devuelve el puntero al principio
if (p_in == p_out)    //Si el puntero de lectura alcanza al de escritura…
error_rebose();       //…se avisa de error de rebose
}
```

```
char *migets(char *ps)//Función de lectura de

{

int i;

char *psc;



i = 0;

psc = ps;

while (1) {

if (p_in != p_out) {

*ps++ = *p_out++;

i++;

}



                if(p_out >= buffer + SMAX)

                      p_out = buffer;



                if (i < MINCHAR)

                        continue;

                else

                        break;

        }

        *ps=0;
```

```
        return psc;

}

//Fin del código del buffer circular

//*************************************************************************
*********

// Configuración

#define RXC    7

#define PB2    2



#ifndef _UART_INT_

#ifdef _MEGAAVRDEV_

#define        rx_counter0           (UCSRA & 0x80)

               #define        tx_counter0           ((!UCSRA) & 0x40)

               #define        rx_counter1           (UCSRA & 0x80)

               #define        tx_counter1           ((!UCSRA) & 0x40)

#endif

#endif

//*************************************************************************
*********



//Funciones de inicialización y configuración de periféricos del
microprocesador

void port_init(void)  //Configuración de las funciones y direcciones I/O de
los puertos
```

```
{
PORTA = 0xFF;  DDRA  = 0xFF;  //Todo salidas. Luces inicialmente apagadas.
        PORTB = 0xFF;  DDRB  = 0xB3;  //Todo  salidas  menos  MISO  (PB6)  y
jumpers (con pull-up)
        PORTC = 0xFF;  DDRC  = 0xFF;
        PORTD = 0xFF;  DDRD  = 0xFF;
}


void uart0_init(void) //Configuración de USART
{
  #ifdef _MEGAAVRDEV_
UCSRB = 0x00;
        UCSRA = 0x00;
        UCSRC = 0x86;
        UBRRH = 0x00;
        UBRRL = 0x0C;  //38400 bauds con un xtal de 8.0MHz
        UCSRB = 0x18;  //Todavía  no  se  activa  la  Interrupción  de  Recepción
Completada
  #endif
}


void init_devices(void)      //Configura resto de periféricos y llama a las
dos anteriores.
```

```
{                    //  Se  la  debe  llamar  antes  de  cualquier  otra
operación.

        CLI(); //Deshabilitar interrupciones durante inicialización.

        port_init();
        uart0_init();
        ACSR=0x80;
        SFIOR=0x00;
        TCCR0=0x00;
        TCNT0=0x00;
        OCR0=0x00;

        SEI(); // Habilitar interrupciones
 }
//*************************************************************************
*********

void main (void)
{
                //Variable declarations. Declaración de variables
        FILE *pf;
```

```
	int i=0;

	char sArchivo[12],sNum[3],sNomb[]="vuelo", sTipo[]=".txt";

	char s[SMAX];



	init_devices();



	//Smal glitch on leds. Pequeño parpadeo para comprobar que el programa comienza

	PORTA &= 0xF3; //Encendemos los leds (rojo y verde)

	delay_ms(250);

PORTA |= 0x0C; //apagandos los leds (rojo y verde)



		//SD card initialization. Inicialización de la tarjeta SD

	while (initialize_media()==0)

	{

		//Blink LED while waiting to initialize

		PORTA ^= 0x0C;

		delay_ms(250);

	}



	PORTA = 0xFB; //When initialization finished, green led on. Cuando está lista, encendemos el led verde



	while(1)
```

```
	{
delay_ms(10);		//retraso para filtrar rebotes (método cutre)
		while(PINB & (1<<PB2));	//Esperamos mientras 1 en el jumper de escritura PB2 (jumper no conectado)
		delay_ms(10);

		PORTA &= 0xF7; //Cuando escribimos datos encendemos el led rojo

		p_in = p_out = buffer;//inicializamos los punteros para el buffer circular

//Looking for the first free file. Buscamos el último archivo escrito.
			for(i=1;i<=99;i++)
		{
				strcpy(sArchivo,sNomb);
				itoa(i,sNum);
				strcat(sArchivo,sNum);
				strcat(sArchivo,sTipo);

if((pf=fopen(sArchivo, APPEND))!=NULL)
				fclose(pf);
			else
					break;
```

```
}

                //Creating the file. Creamos el archivo de datos

                pf=fcreate(sArchivo, 0);


                //Writing its name in. Escribimos el nombre del archivo dentro
de él

                fprintf(pf, "%s\r\n", sArchivo);


                UCSRB = 0x98; //Activamos Interrupción de Recepción Completada


                //What is red from the USART, is writed in the file. Lo que
leemos de la vía serie lo imprimimos en el archivo

                while(!(PINB & (1<<PB2)))    //While   the   jumper   remains
connected.

                {

                        fprintf(pf, "%s", migets(s));

                }


                 UCSRB  =  0x18;  //desactivamos  interrupción  de  recepción
completa


                       fflush(pf);
```

```
                fclose(pf);



                PORTA |= 0x08; //When writing finished, red led off. Cuando
acabamos de escribir datos, apagamos el led rojo

        }

}
```

ANEXO III: CÓDIGO MODIFICADO DE LA UNIDAD DE MEDIDA INERCIAL

Los cambios realizados respecto de [3] han sido resaltados en rojo y explicados en el apartado 3.2

Mainloop.c

```
/* -*- indent-tabs-mode:T; c-basic-offset:8; tab-width:8; -*- vi: set ts=8:

 * $Id: mainloop.c,v 2.26 2003/07/13 17:15:43 tramm Exp $

 *

 * Mainloop for the rev2.x boards.  Integrates the ADC, UART, NMEA and

 * tachometer, as well as lots of other stuff.

 *

 * (c) 2002 Trammell Hudson <hudson@swcp.com>

 *

 *************

 *

 *  This file is part of the autopilot onboard code package.

 *

 *  Autopilot is free software; you can redistribute it and/or modify

 *  it under the terms of the GNU General Public License as published by
```

```
 *  the Free Software Foundation; either version 2 of the License, or

 *  (at your option) any later version.

 *

 *  Autopilot is distributed in the hope that it will be useful,

 *  but WITHOUT ANY WARRANTY; without even the implied warranty of

 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the

 *  GNU General Public License for more details.

 *

 *  You should have received a copy of the GNU General Public License

 *  along with Autopilot; if not, write to the Free Software

 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA   02111-1307
USA

 *

 */




#include <avr/io.h>

#include <avr/pgmspace.h>



#include "timer.h"

#include "uart.h"

#include "string.h"
```

```
#include "servo.h"

#include "ppm.h"

#include "adc.h"


//#define CONFIG_PNI


#ifdef CONFIG_V2X

#include "v2x.h"

#endif


#ifdef CONFIG_PNI

#include "pni.h"

#endif




/**

 *  These are optional features that control the software UART (for

 * a GPS), the engine tach pulse counter and the LCD output.  We're not

 * using them right now since they are detracting from the main goal

 * of servo driving and flight control

 */

#if 0

#include "soft_uart.h"
```

```
#include "tach.h"

#include "lcd.h"

#include "led.h"

#endif




static uint8_t        manual;

static uint8_t        mode;

static uint16_t              command[8];




/*

 * Command processing and servo manipulation happens in here

 * We are only called by the PPM interrupt once a complete frame is

 * ready for us, so there is no need to check that things are good.

 *

 * The servo outputs are in a set order, regardless of which radio

 * type is used.  They are:

 *

 *     0      Unused

 *     1      Throttle

 *     2      Roll

 *     3      Pitch
```

```
 *      4       Yaw
 *      5       Manual/Auto switch (not connected)
 *      6       Collective
 *      7       Gyro gain
 *      8       Extra (unused)
 *      9       Mode switch (not connected)
 *
 * The high speed servo output is just a copy of the tail servo (4).
 */
void
copy_to_servos( void )
{
        /*
         * Toggle the manual control flag when ever the
         * H thingy is flipped.  This is a vital thing
         * to happen!
         */
        uint16_t mode_width = ppm_pulses[ mode_index ];

        mode =
                mode_width < mode_threshold_0 * CLOCK ? 0 :
                mode_width < mode_threshold_1 * CLOCK ? 1 :
                2;
```

```
manual = ppm_pulses[ manual_index ] > manual_threshold * CLOCK;



/* Manual means all servos go to the receiver */
if( manual )
{
        /*
         * Copy the current frame to our command so that we do not
         * jump at all when switching to automatic mode
         */
        command[0] = ppm_pulses[ roll_index ];
        command[1] = ppm_pulses[ pitch_index ];
        command[2] = ppm_pulses[ yaw_index ];
        command[3] = ppm_pulses[ collective_index ];
        command[4] = ppm_pulses[ throttle_index ];
}


/* Determine who controls which servos */
const uint8_t  manual_roll    = manual;
const uint8_t  manual_pitch   = manual || mode < 1;
const uint8_t  manual_yaw     = manual || mode < 1;
const uint8_t  manual_coll    = manual || mode < 2;
```

```
const uint8_t  manual_throttle       = manual || mode < 2;


servo_widths[1] = manual_throttle
       ? ppm_pulses[ throttle_index ]
       : command[4];


/*
 * Mechanical mixing.  Drive the roll, pitch and collective
 * servo directly from the radio or computer generated signals.
 */
servo_widths[2] = manual_roll
       ? ppm_pulses[ roll_index ]
       : command[0];


servo_widths[3] = manual_pitch
       ? ppm_pulses[ pitch_index ]
       : command[1];


/* This channel is unused */
servo_widths[5] = ppm_pulses[ manual_index ];


servo_widths[6] = manual_coll
       ? ppm_pulses[ collective_index ]
```

```
                : command[3];


        servo_widths[4] = manual_yaw
                ? ppm_pulses[ yaw_index ]
                : command[2];


        /* We should force normal mode on the gyro here */
        servo_widths[7] = ppm_pulses[ yaw_gain_index ];


        /* Extra servo. */
        servo_widths[8] = ppm_pulses[ extra_index ];


        /* This channel is unused */
        servo_widths[9] = ppm_pulses[ mode_index ];


        /* Our high speed servo is a copy of our tail servo */
        servo_hs = servo_widths[ 4 ];
}



static inline void
input_init( void )
{
```

```
        uint8_t               i;


        for( i=0 ; i<8 ; i++ )
                command[i] = servo_make_pulse_width( 32768ul );
}




static inline void
reset( void )
{
        void (*bootloader)( void ) = (void*) BOOT_BASE;


        /* No interrupts until we restart in the boot loader */
        cli();


        /* No timer interrupts */
        TIMSK = 0;


        /* No OCR toggles */
        TCCR1A = 0;


        bootloader();
}
```

```
SIGNAL( SIG_UART_RECV )

{

        static uint8_t        phase;

        static uint8_t        servo;

        static uint8_t        high_bits;

        static uint8_t        low_bits;


        unsigned char         c = inp( UDR );


                switch( phase )

                {

                case 0:

                        /* Wait for sync pulse */

                        if( c == 0xFF )

                                phase = 1;

                        break;


                case 1:

                        servo = c;

                        phase = 2;

                        break;
```

```
case 2:
        high_bits = c;
        phase = 3;
        break;

case 3:
        low_bits = c;

        /* Check for reset command */
        if( servo     == 0xDE
        &&  high_bits  == 0xAD
        &&  low_bits   == 0xBE
        )
                reset();

        if( servo > SERVO_MAX )
                break;

        command[ servo ] = 0
                | ((int)low_bits)  << 0
                | ((int)high_bits) << 8;
```

```
/*
                        puts( "$SRV" );
                        put_uint8_t( servo );
                        putc( ',' );
                        put_uint16_t( command[ servo ] );
                        putnl();
*/

                        /* Fall through */

                default:
                        phase = 0;
                        break;
                }
}


#ifdef CONFIG_PNI
static inline void
pni_output( void )
{
        if( pni_state != 0 )
                return;
```

```
        if( pni_valid > 0 )
        {
                puts( "$GPHDG," );
                put_uint16_t( pni_values[0] );
                putc( ',' );
                put_uint16_t( pni_values[1] );
                putc( ',' );
                put_uint16_t( pni_values[2] );
                putnl();
                pni_valid = 0;
        }

        if( pni_valid == 0 )
        {
                if( pni_read_axis() == 0 )
                        return;
        } else {
                pni_valid = 0;
        }

        puts( "$ERPNI\r\n" );
}
```

```
#endif



int main( void )
{
        timer_init();
        uart_init();
        servo_init();
        ppm_init();
        adc_init();

#ifdef CONFIG_PNI
        pni_init();
#endif

        input_init();

        sei();

        puts( "$Id: mainloop.c,v 2.26 2003/07/13 17:15:43 tramm Exp $\r\n" );

        while( 1 )
        {
```

```
                //input_task();
                //user_task();

                /* Every 32768 microseconds */
                if( timer_periodic() == 0 )
                        continue;

                adc_output();

                if( ppm_valid )
                {
                        ppm_output();
                        ppm_valid = 0;
                }

#ifdef CONFIG_PNI
                pni_output();
#endif
        }
}
```

Config.h

```
/*
 * $Id: config.h,v 1.1 2003/02/20 16:53:52 tramm Exp $
 *
 * Configuration file for the different feature sets and
 * pin outs
 */

#ifndef _REV2_CONFIG_H_
#define _REV2_CONFIG_H_

#define VERSION              2.4

/*************************************************************************
 *
 * Feature selection
 *
 */

/*
 * Low speed servo banks driven by the 16 bit timer overflows.
 * Uses a 4017 connected to the OCRnw pins and a reset pin.
```

```
 */
#define SERVO_BANK_A
//#define SERVO_BANK_B



/*
 * High speed servos driven by the 16 bit timer overflows.
 * Directly connects to the outputs.  No reset pin necessary
 */
//#define SERVO_HS_A
//#define SERVO_HS_B



/*
 * PPM / PCM decoding is selectable, although PCM decoding is
 * very buggy right now.
 */
#define PCM_INPUT
#define PPM_INPUT



/*************************************************************************
 *
```

```
 * Consistency checks
 *
 */

#if defined( SERVO_HS_A ) && defined( SERVO_BANK_A )
#error "Only one of SERVO_HS_A and SERVO_BANK_A may be defined"
#endif

#if defined( SERVO_HS_B ) && defined( SERVO_BANK_B )
#error "Only one of SERVO_HS_B and SERVO_BANK_B may be defined"
#endif

#if defined( PCM_INPUT ) && defined( PPM_INPUT )
#error "Only one of PCM_INPUT and PPM_INPUT may be defined"
#endif



/*************************************************************************
 *
 * Board revision specific configurations, with pin assignments
 *
 */
```

```
#if VERSION == 2.2

/**

 *  2.2 board specific configurations

 */




#elif VERSION == 2.4



/**

 *  2.4 board specific configurations

 */

#ifdef SERVO_BANK_B

#error "2.4 has no servo bank B"

#endif



#else

#error "Unknown board revision"

#endif



#endif
```

Ppm.h

```
/* -*- indent-tabs-mode:T; c-basic-offset:8; tab-width:8; -*- vi: set ts=8:

 * $Id: ppm.h,v 2.10 2003/07/08 17:57:29 tramm Exp $

 *

 * Decoder for the trainer ports or hacked receivers for both

 * Futaba and JR formats.  The ppm_valid flag is set whenever

 * a valid frame is received.

 *

 * Pulse widths are stored as unscaled 16-bit values in ppm_pulses[].

 * If you require actual microsecond values, divide by CLOCK.

 * For an 8 Mhz clock and typical servo values, these will range

 * from 0x1F00 to 0x4000.

 *

 * (c) 2002 Trammell Hudson <hudson@rotomotion.com>

 *************

 *

 *  This file is part of the autopilot onboard code package.

 *

 *  Autopilot is free software; you can redistribute it and/or modify

 *  it under the terms of the GNU General Public License as published by

 *  the Free Software Foundation; either version 2 of the License, or

 *  (at your option) any later version.
```

```
 *
 *  Autopilot is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with Autopilot; if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA   02111-1307
USA
 *
 */


#ifndef _ppm_h_
#define _ppm_h_


/**
 *  Receiver types are:
 *    1  Futaba
 *    2  JR
 */
#define PPM_RX_TYPE   1
```

```
#include <string.h>          /* For memset */
#include "string.h"          /* For puts, etc */
#include "uart.h"



#if PPM_RX_TYPE == 1


/*
 * Futaba assignments for Trammell's radio on the Concept.
 * Falling edge detect.  Thresholds are in microseconds.
 */
#       define  PPM_MAX_PULSES        6
#       define  EDGE_DETECT           cbi


#       define  collective_index      5
#       define  throttle_index        2
#       define  roll_index            0
#       define  pitch_index           1
#       define  yaw_index             3
#       define  yaw_gain_index        6
#       define  extra_index           7
```

```
#       define  manual_index          4
#       define  manual_threshold      1500


#       define  mode_index            6
#       define  mode_threshold_0      1000
#       define  mode_threshold_1      1800


#elif PPM_RX_TYPE == 2


/*
 * JR assignments for Dennis' radio on the Nova.
 * Rising edge detect.
 */
#       define  PPM_MAX_PULSES        6
#       define  EDGE_DETECT           sbi


#       define  collective_index      5
#       define  throttle_index        0
#       define  roll_index            1
#       define  pitch_index           2
#       define  yaw_index             3
#       define  yaw_gain_index        6
#       define  extra_index           8
```

```
#       define  manual_index          4
#       define  manual_threshold      1500

#       define  mode_index            8
#       define  mode_threshold_0      1300
#       define  mode_threshold_1      1600

#else

#       error   "Unknown PPM type!"
#endif


/*
 * User routine to copy (and modify) the PPM values to the servos
 * as soon as they are ready.  This should be as fast as possible.
 */
static void copy_to_servos( void );


/*
 * PPM pulses are falling edge clocked on the ICP, which records
```

```
 * the state of the global clock.  We do not use any noise
 * canceling features.
 *
 * JR might be rising edge clocked; set that as an option
 */
static inline void
ppm_init( void )
{
        EDGE_DETECT( TCCR1B, ICES1 );

        /* No noise cancelation */
        cbi( TCCR1B, ICNC1 );

        /* Set ICP to input, internal pull up */
        sbi( PORTD, 6 );
        cbi( DDRD, 6 );

        /* Enable interrupt on input capture */
        sbi( TIMSK, TICIE1 );
}

/*
```

```
 *  The pulse widths are recorded into this array.  If the values
 * are not zeroed by the users, they will never be reset.
 */
uint16_t              ppm_pulses[ PPM_MAX_PULSES ];
uint8_t               ppm_valid;



/*
 * Pulse width is computed as the difference between now and the
 * previous pulse.  If no pulse has been received between then and
 * now, the time of the last pulse will be equal to the last pulse
 * we measured.  Unfortunately, the Input Capture Flag (ICF1) will
 * not be set since the interrupt routine disables it.
 *
 * Sync pulses are timed with Timer2, which runs at Clk/1024.  This
 * is slow enough at both 4 and 8 Mhz to measure the lengthy (10ms
 * or longer) pulse.
 *
 * Otherwise, compute the pulse width with the 16-bit timer1,
 * push the pulse width onto the stack and increment the
 * pulse counter until we have received eight pulses.
 */
SIGNAL( SIG_INPUT_CAPTURE1 )
```

```
{
        static uint16_t              last;
        uint16_t             this;
        uint16_t             width;
        static uint8_t       state;
        static uint8_t       sync_start; // Also sync width

        this          = ICR1;
        width         = this - last;
        last          = this;

        if( state == 0 )
        {
                uint8_t              end    = inp( TCNT2 );
                uint8_t              diff   = end - sync_start;
                sync_start = end;

                /*
                 * Waiting for a sync pulse at least 10 ms long.
                 * 10000 us * CLOCK / 1024 => about 9 * CLOCK ticks.
                 * If we use Clk/256 then it overflows too quickly.
                 */
                if( diff > 4 * CLOCK )
```

```
        {
                state = 1;
                sync_start = diff;
        }

        return;
}

if( state < PPM_MAX_PULSES+1 )
{
        /* Sanity check the pulse for a valid servo width */
        if( width < 700ul * CLOCK || width > 2300ul * CLOCK )
                goto restart_ppm_cycle;

        /* Read a data pulses */
        ppm_pulses[ state++ - 1 ] = width;
        return;
}

/*
 * All done.  Mark it as valid and call the user supplied
 * copy_to_servos() routine to update the servo outputs.
 */
```

```
        ppm_valid      = 1;
        copy_to_servos();

restart_ppm_cycle:
        state          = 0;
        sync_start     = TCNT2;
}


static inline void
ppm_output( void )
{
        uint8_t               i;

        puts( "$GPPPM" );

        if( !ppm_valid )
        {
                puts( ",,,,,,,,\r\n" );
                return;
        }

        for( i=0 ; i < PPM_MAX_PULSES ; i++ )
```

```
        {
                putc( ',' );
                put_uint16_t( ppm_pulses[i] );
        }

        putnl();
}

#endif
```

Servo.c

```
/* -*- indent-tabs-mode:T; c-basic-offset:8; tab-width:8; -*- vi: set ts=8:
 * $Id: servo.c,v 1.4 2003/06/24 22:50:34 tramm Exp $
 *
 * (c) 2002 Trammell Hudson <hudson@rotomotion.com>
 *
 *************
 *
 *  This file is part of the autopilot onboard code package.
 *
```

```
 *  Autopilot is free software; you can redistribute it and/or modify

 *  it under the terms of the GNU General Public License as published by

 *  the Free Software Foundation; either version 2 of the License, or

 *  (at your option) any later version.

 *

 *  Autopilot is distributed in the hope that it will be useful,

 *  but WITHOUT ANY WARRANTY; without even the implied warranty of

 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the

 *  GNU General Public License for more details.

 *

 *  You should have received a copy of the GNU General Public License

 *  along with Autopilot; if not, write to the Free Software

 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA   02111-1307
USA

 *

 */



#include <avr/io.h>

#include <avr/signal.h>

#include "servo.h"

#include "timer.h"



uint16_t              servo_widths[ SERVO_MAX ];
```

```
uint16_t              servo_hs;


/* Current servo being driven */

static uint8_t        servo;


/* High speed servo state */

static uint8_t               hs_state;









/*

 * For the 2.4 boards, Servo bank A is the only 4017 bank and

 * it is driven by OCR1A with reset on PORTD7.

 */

#define SERVO_PORT           PORTD

#define SERVO_DDR            DDRD


#define SERVO_A_OCR          OCR1A

#define SERVO_A_ENABLE       OCIE1A

#define SERVO_A_FLAG         OCF1A

#define SERVO_A_FORCE        FOC1A

#define SERVO_A_RESET        7
```

```
#define SERVO_A_CLOCK        5

#define SERVO_A_COM0         COM1A0

#define SERVO_A_COM1         COM1A1



#define SERVO_HS_OCR         OCR1B

#define SERVO_HS_ENABLE              OCIE1B

#define SERVO_HS_FLAG        OCF1B

#define SERVO_HS_CLOCK       4

#define SERVO_HS_COM0        COM1B0

#define SERVO_HS_COM1        COM1B1




/*

 * For the High speed servo output, you must select either CONFIG_FUTABA

 * or CONFIG_JR to drive the digital servos.  From experimental tests,

 * the rates are:

 *

 *  Futaba    275 Hz == 3.65 ms frame rate

 *  JR        166 Hz == 6.00 ms frame rate

 */

#define CONFIG_FUTABA



#if defined( CONFIG_FUTABA )
```

```
# define HS_SPEED            3650 * CLOCK

#elif defined( CONFIG_JR )

# define HS_SPEED            6000 * CLOCK

#else

# error "servo.c: Neither FUTABA nor JR is defined for high speed output"

#endif
```

```
/*

 *  We use the output compare registers to generate our servo pulses.

 * These should be connected to a decade counter that routes the

 * pulses to the appropriate servo.

 *

 * Initialization involves:

 *

 * - Reseting the decade counters

 * - Writing the first pulse width to the counters

 * - Setting output compare to set the clock line by calling servo_enable()

 * - Bringing down the reset lines

 *

 * Ideally, you can use two decade counters to drive 20 servos.

 */
```

```
void

servo_init( void )

{

        uint8_t               i;

        uint16_t              mid_point = servo_make_pulse_width( 32768 );



        /* Configure the reset and clock lines */

        SERVO_DDR |= 0

                | ( 1 << SERVO_A_RESET )

                | ( 1 << SERVO_A_CLOCK )

                | ( 1 << SERVO_HS_CLOCK );



        /* Reset the decade counter */

        sbi( SERVO_PORT, SERVO_A_RESET );



        /* Lower the HS servo line */

        cbi( SERVO_PORT, SERVO_HS_CLOCK );



        /* Lower the regular servo line */

        cbi( SERVO_PORT, SERVO_A_CLOCK );



        /* Set all servos (including HS) at their midpoints */

        servo_hs = mid_point;
```

```
for( i=0 ; i < SERVO_MAX ; i++ )
        servo_widths[i] = mid_point;

/* Select the first servo */
servo          = 1;

/* Reset the high speed servo driver */
hs_state       = 0;

/* Set the regular servos to go off some long time from now */
SERVO_A_OCR    = 32768ul;
SERVO_HS_OCR   = 65535ul;

/*
 * Configure output compare to toggle the output bits.
 * The high speed servo alternates between the on and
 * off state, while the regular servo will force a
 * change.
 */
TCCR1A |= 0
        | ( 1 << SERVO_HS_COM0 )
        | ( 0 << SERVO_HS_COM1 )
```

```
                | ( 1 << SERVO_A_COM0 )
                | ( 0 << SERVO_A_COM1 );

        /* Clear the interrupt flags in case they are set */
        TIFR = 0
                | ( 1 << SERVO_HS_FLAG )
                | ( 1 << SERVO_A_FLAG );

        /* Unassert the decade counter reset to start it running */
        cbi( SERVO_PORT, SERVO_A_RESET );

        /* Enable our output compare interrupts */
        TIMSK |= 0
                | ( 1 << SERVO_A_ENABLE )
                | ( 1 << SERVO_HS_ENABLE );
}

/*
 * For each servo output, we check to see if the output compare flag
 * has been set.  If it has been set, we compute the next pulse width
 * and reset the output value.
 */
```

```
SIGNAL( SIG_OUTPUT_COMPARE1A )

{

        uint16_t              width;

        uint8_t               tmp = servo;



        if( tmp < SERVO_MAX - 2 )

                goto no_reset;

        sbi( SERVO_PORT, SERVO_A_RESET );

        tmp = -1;



        no_reset:

        servo = ++tmp;

        width = servo_widths[ tmp ];



        if( width < 1000ul * CLOCK )

                width = 1000ul * CLOCK;



        if( tmp == 0 )

                cbi( SERVO_PORT, SERVO_A_RESET );



        SERVO_A_OCR += width;



        /* Force a comparison to toggle the clock pin */
```

```
        TCCR1A |= ( 1 << SERVO_A_FORCE );
}




/*
 * High speed servo routine.
 *
 * We want to toggle between on for servo_hs time and off for
 * HS_SPEED (either 275 Hz == 3.65 ms for Futaba or 166 Hz == 6 ms
 * for JR).
 */
SIGNAL( SIG_OUTPUT_COMPARE1B )
{
        if( hs_state )
        {
                hs_state = 0;
                SERVO_HS_OCR += HS_SPEED - servo_hs;
        } else {
                hs_state = 1;
                SERVO_HS_OCR += servo_hs;
        }
}
```

Makefile

```
#!/usr/bin/make

# $Id: Makefile,v 2.13 2003/06/24 22:54:15 tramm Exp $

#

# Rev2 control board and IMU makefile

#



ARCH           = atmega32



atmega163_base = 0x3C00

atmega163_clock       = 8



atmega16_base  = 0x3C00

atmega16_clock = 16



atmega32_base  = 0x7C00

atmega32_clock = 16







CC             = /usr/bin/avr-gcc -mmcu=$(ARCH)

LD             = $(CC)

SIZE           = avr-size
```

```
SERIAL_FLAGS   =                                                \
        -dprog=avr910                                           \
        -dpart=auto                                             \
        -dserial=/dev/ttyS0                                     \
        -dspeed=38400                                           \

ISP_FLAGS      =                                                \
        -dprog=stk200                                           \

UISP           = /usr/bin/uisp -v
#UISP_FLAGS    = $(ISP_FLAGS)
UISP_FLAGS     = $(SERIAL_FLAGS)

CFLAGS         =                                                \
        -O4                                                     \
        -W                                                      \
        -Wall                                                   \
        -DBOOT_BASE=$($(ARCH)_base)                             \
        -DCLOCK_SPEED=$($(ARCH)_clock)                                  \

LDFLAGS        =                                                \
```

```
        -lm                                                     \


all:                                                            \
        loader.srec                                             \
        mainloop.srec                                           \


install: mainloop.install



mainloop.srcs  =                                                \
        mainloop.c                                              \
        uart.c                                                  \
        string.c                                                \
        adc.c                                                   \
        servo.c                                                 \


AHRS_EXTRA     =                                                \
        ahrs.c                                                  \
        imu.c                                                   \
        pid.c                                                   \
        calib.c                                                 \
        mat.c                                                   \
```

```
NO=\

        soft_uart.c                                             \

        bob.c                                                   \

        lcd.c                                                   \




mainloop.objs  =                                                \

        $(mainloop.srcs:.c=.o)                                  \


mainloop.elf: $(mainloop.objs)




pcm.srcs       =                                                \

        pcm.c                                                   \

        string.c                                                \

        uart.c                                                  \


pcm.objs       =                                                \

        $(pcm.srcs:.c=.o)                                       \


pcm.elf: $(pcm.objs)
```

```
#
# Boot loader is always installed over the ISP
#
loader.srcs    =                                                \
        loader.c                                                \
        assembly.S                                              \

loader.objs    =                                                \
        $(foreach a,$(loader.srcs),$(basename $a).o)            \

loader.elf: $(loader.objs)
        $(LD)                                                   \
                $(LDFLAGS)                                      \
                $($(@:.elf=).objs)                              \
                -Ttext $($(ARCH)_base)                          \
                -o $@                                           \

        $(SIZE) $@

loader.install: loader.srec erase
        $(UISP) $(ISP_FLAGS) --upload if="$<"
```

```
#
# General rules
#
%.elf:
	$(LD)                                                   \
		$($(@:.elf=).objs)                              \
		-o $@                                           \
		$(LDFLAGS)                                      \

	$(SIZE) $@


%.srec: %.elf
	avr-objcopy -O srec $< $@


%.install: %.srec
	# stk200 needs to be erased first
	$(UISP) $(SERIAL_FLAGS) --erase
	$(UISP) $(SERIAL_FLAGS) --upload if=$<


erase:
	$(UISP) $(ISP_FLAGS) --erase
```

```
clean:
	rm -f *.srec *.elf a.out core *.o *.a



#
# Dependencies
#
mainloop.o:                                                    \
	mainloop.c                                             \
	timer.h                                                \
	uart.h                                                 \
	string.h                                               \
	soft_uart.h                                            \
	servo.h                                                \
	ppm.h                                                  \
	adc.h                                                  \
	led.h                                                  \
	tach.h                                                 \
	button.h                                               \


adc.o:                                                         \
	adc.c                                                  \
```

```
        adc.h                                                   \
        average.h                                               \


string.o:                                                       \
        string.c                                                \
        string.h                                                \
        uart.h                                                  \


uart.o:                                                         \
        uart.c                                                  \
        uart.h                                                  \


lcd.o:                                                          \
        lcd.c                                                   \
        lcd.h                                                   \


loader.o:                                                       \
        loader.c                                                \
        led.h                                                   \
        button.h                                                \


soft_uart.o:                                                    \
        soft_uart.c                                             \
```

```
soft_uart.h                                                    \

string.h                                                       \
```

ANEXO IV: CARACTERÍSTICAS DEL CONECTOR/ZÓCALO PARA TARJETAS SD

En este anexo se incluye un plano, tanto dimensional como funcional, del zócalo para tarjetas SD utilizado en la construcción del Sistema de Adquisición de Datos. Dicho zócalo es el modelo S01V01381 de la casa JAE Ltd –Jaran Aviation Electronics-.

En la esquina inferior derecha se encuentra:

- El cajetín de identificación del plano.
- Una tabla con los diferentes contactos del zócalo explicando su función, identificando el material con que están hechos y mostrando algunos comentarios.
- Un cuadro con los dos posibles modelos de fijación que se tienen disponibles.

En la esquina inferior izquierda se encuentra una tabla explicativa de los posibles estados de los contactos según se haya introducido la tarjeta o no y según esté o no protegida contra escritura.

El resto de la parte inferior está ocupada por una serie de notas sobre el plano y su interpretación y el resto del plano contiene las diferentes vistas del zócalo convenientemente acotadas.

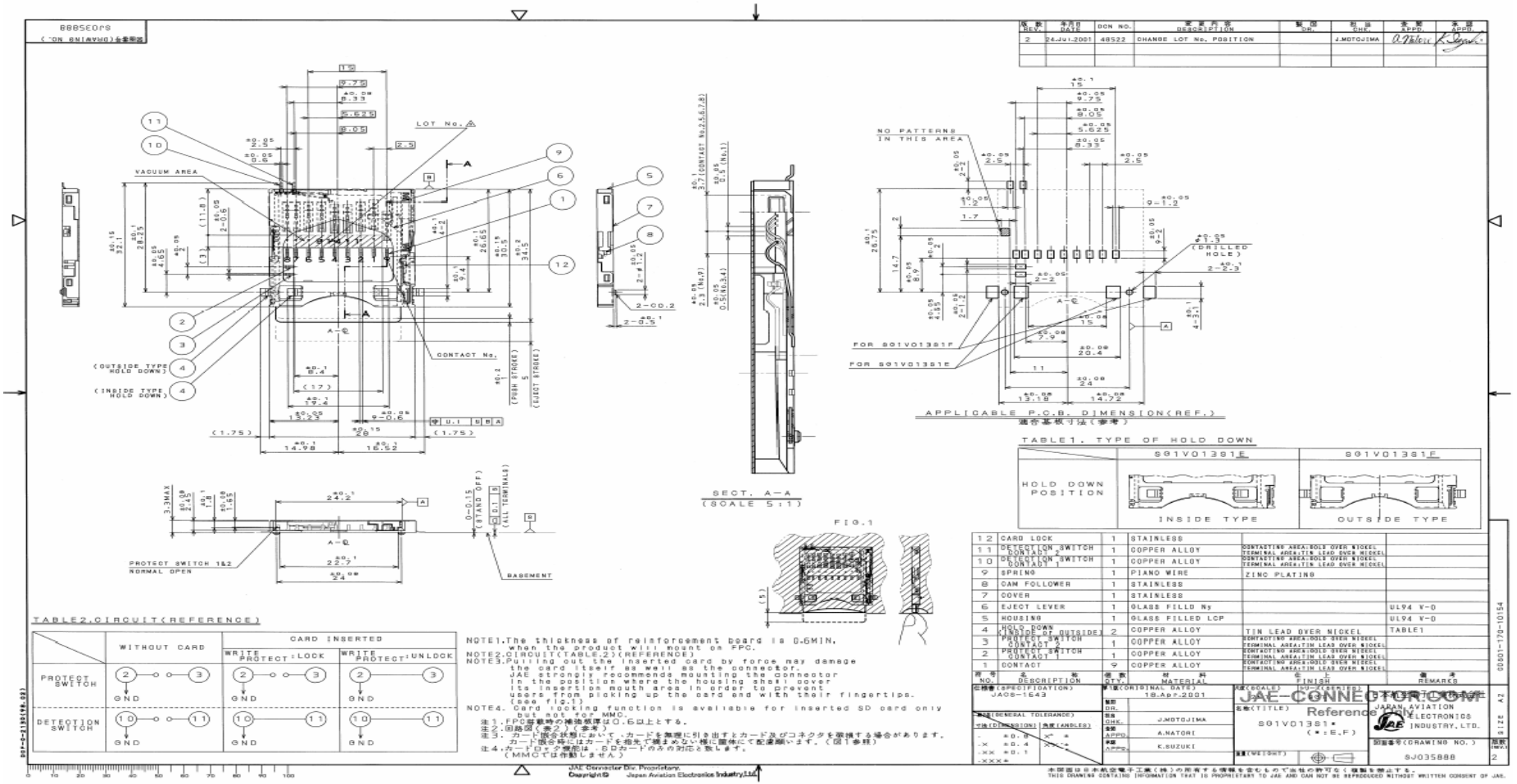


TABLE1. TYPE OF HOLD DOWN

HOLD DOWN POSITION	S01V01381E	S01V01381F
	INSIDE TYPE	OUTSIDE TYPE

TABLE2.CIRCUIT(REFERENCE)

	WITHOUT CARD	CARD INSERTED WRITE PROTECT:LOCK	CARD INSERTED WRITE PROTECT:UNLOCK
PROTECT SWITCH	2 — 3, 2→GND	2 — 3, 2→GND	2 — 3, 2→GND
DETECTION SWITCH	10 — 11, 10→GND	10 — 11, 10→GND	10 — 11, 10→GND

NOTE1.The thickness of reinforcement board is 0.6MIN. when the product will mount on FPC.
NOTE2.CIRCUIT(TABLE.2)(REFERENCE)
NOTE3.Pulling out the inserted card by force may damage the card itself as well as the connector. JAE strongly recommends mounting the connector in the position where the housing shall cover its insertion mouth area in order to prevent users from picking up the card end with their fingertips. (see fig.1)
NOTE4. Card locking function is available for inserted SD card only but not for MMC.
注1.FPC基板時の補強板厚は0.6以上とする。
注2.回路図(参2)(参考)
注3. カード脱合状態において、カードを無理に引き出すとカード及びコネクタを破損する場合があります。カード脱合時にはカードを指先で摘まめない様に筐体にて配慮願います。(図1参照)
注4.カードロック機能は、SDカードのみの対応と致します。(MMCでは作動しません)

NO.	DESCRIPTION	QTY.	MATERIAL	FINISH	REMARKS
12	CARD LOCK	1	STAINLESS		
11	DETECTION SWITCH CONTACT 2	1	COPPER ALLOY	CONTACTING AREA:GOLD OVER NICKEL TERMINAL AREA:TIN LEAD OVER NICKEL	
10	DETECTION SWITCH CONTACT 1	1	COPPER ALLOY	CONTACTING AREA:GOLD OVER NICKEL TERMINAL AREA:TIN LEAD OVER NICKEL	
9	SPRING	1	PIANO WIRE	ZINC PLATING	
8	CAM FOLLOWER	1	STAINLESS		
7	COVER	1	STAINLESS		
6	EJECT LEVER	1	GLASS FILLD Ny		UL94 V-0
5	HOUSING	1	GLASS FILLED LCP		UL94 V-0
4	HOLD DOWN (INSIDE OR OUTSIDE)	2	COPPER ALLOY	TIN LEAD OVER NICKEL	TABLE1
3	PROTECT SWITCH CONTACT 2	1	COPPER ALLOY	CONTACTING AREA:GOLD OVER NICKEL TERMINAL AREA:TIN LEAD OVER NICKEL	
2	PROTECT SWITCH CONTACT 1	1	COPPER ALLOY	CONTACTING AREA:GOLD OVER NICKEL TERMINAL AREA:TIN LEAD OVER NICKEL	
1	CONTACT	9	COPPER ALLOY	CONTACTING AREA:GOLD OVER NICKEL TERMINAL AREA:TIN LEAD OVER NICKEL	

仕様書(SPECIFICATION)	JAQS-1643		
製図(ORIGINAL DATE)	18.Apr.2001	尺度(SCALE)	NU-ZIGENIEI
一般公差(GENERAL TOLERANCE)		名称(TITLE)	S01V01381* (*:E,F)
寸法(DIMENSION)	角度(ANGLES)	製図 DR.	J.MOTOJIMA
- ±0.8	° ±	照合 CHK.	A.NATORI
.X ±0.4	X° ±	承認 APPD.	K.SUZUKI
.XX ±0.1	X.X° ±	質量(WEIGHT)	
.XXX±			

JAE-CONNECTOR.COM
Reference Only
JAPAN AVIATION ELECTRONICS INDUSTRY, LTD.
図面番号(DRAWING NO.) SJ035888
改訂(REV.) 2
SIZE A2

REV.	改訂日 DATE	DCN NO.	変更内容 DESCRIPTION	製図 DR.	照合 CHK.	承認 APPD.	承認 APPD.
2	24-JUL-2001	48522	CHANGE LOT No. POSITION		J.MOTOJIMA	A.Natori	K.Suzuki

JAE Connector Div. Proprietary.
Copyright© Japan Aviation Electronics Industry,Ltd.
本図面は日本航空電子工業(株)の所有する情報を含むもので当社の許可なく複製を禁止する。
THIS DRAWING CONTAINS INFORMATION THAT IS PROPRIETARY TO JAE AND CAN NOT BE REPRODUCED WITHOUT WRITTEN CONSENT OF JAE.

Parte II Estudio Económico

Capítulo 1 COSTE DEL PROYECTO

En este capítulo se desglosan los costes que en los que hubiera incurrido una empresa en el caso de haber acometido esta parte proyecto desde cero.

El gasto real no ha sido tanto por los siguientes motivos:

- El sueldo de ingeniero que aquí se menciona es nulo por ser este un Proyecto Fin de Carrera llevado a cabo por el que escribe.
- El ordenador que aquí se incluye como coste es propiedad del proyectando.
- Las licencias del software empleado son compartidas por todos los alumnos y profesores de la Universidad.

Desglose de costes (en €uros)

- Construcción de prototipos
  - IMU 353
    - Compra a Rotomotion 325
    - Baterías LIPO 1250mAh 28
  - SAD 310
    - Compra Flash File 160
    - Fabricación placas 60
    - Componentes 90
  - Caja para montaje IMU + SAD 9,5
- Sueldo Ingeniero (700 horas) 8945
- Consumo eléctrico 25
- Ordenador 1479
- Licencias software 948
  - Code Vision (HP-InfoTech) 150
  - Eagle v4.11 Standart (CadSoft) 798
  - AVR Studio 4 (ATMEL) 0
  - AVR Dude 0
  - Kargador 0
- Recursos Aeromodelísticos 49
  - Micro-Receptor ACT 49

- TOTAL 12118,

Parte III MANUAL DE USUARIO

La Ilustración 40 muestra la integración de los módulos IMU y SAD. Como podemos observar la parte frontal es un panel de mandos constituido por:

- Interruptor de alimentación, con dos posiciones: “ON” y “OFF”.
- Interruptor de escritura, con dos posiciones: “REC” y “STOP”.
- Ranura para introducción/extracción de la tarjeta SD, marcada como “SD”.
- Led rojo marcado como “STATUS”.
- Led verde marcado como “POWER”.

Ilustración 40 – Integración IMU+SAD

La interfaz de usuario para utilizar el IMU+SAD es muy sencilla, se han de seguir los siguientes pasos:

1. Conectar la vía serie de un sistema que trabaje a 38400 baudios con la entrada serie del SAD. En nuestro caso la IMU ya está conectada.
2. Si se quiere conectar algún sistema que trabaje a otra velocidad de transmisión habrá que modificar el código de programa. En la función *void uart_init(void)* se debería elegir la velocidad deseada modificando UBRRL según el "Anexo I: Guía básica sobre manejo de AVR ATMEL Atmega".
3. Poner en ON el interruptor de alimentación.
4. Tras la inicialización de los periféricos un pequeño parpadeo aparece en los leds, dando paso a la inicialización de la tarjeta SD.
5. Mientras se inicializa la tarjeta SD continua ese parpadeo y si la inicialización es correcta el led verde se queda encendido y el rojo apagado. En caso contrario los leds se quedan encendidos y se debe comprobar, por este orden, si:
    a. La tarjeta SD está debidamente introducida en la ranura correspondiente (en nuestro caso, marcada como "SD").
    b. La tarjeta es la adecuada (capacidad menor de 64 MB)
    c. La tarjeta esta correctamente formateada (en nuestro caso, FAT12).
6. Poner en REC el interruptor de escritura, con lo que se empieza a guardar datos en la SD.
7. El led rojo se enciende cuando las operaciones de escritura se están llevando a cabo.
8. Cuando el usuario quiera dar por terminada la sesión de escritura de datos tan solo tendrá que colocar el interruptor en la posición STOP, con lo que la luz roja se apagará.

9. Si se quiere comenzar otro archivo seguir los pasos 6, 7 y 8.
10. Repetir el paso 9 tantas veces como archivos de datos quieran hacer.
11. Para apagar el sistema llevar el interruptor de alimentación a su posición OFF.
12. Ahora la SD puede extraerse de su ranura y manipular sus archivos con un lector comercial. En nuestro caso, los archivos vienen nombrados como vuelo1.txt, vuelo2.txt, etc, si esta nomenclatura se quisiera cambiar se debería modificar en la función main la declaración *sNomb[]=”vuelo”* por el nombre que se crea adecuado.